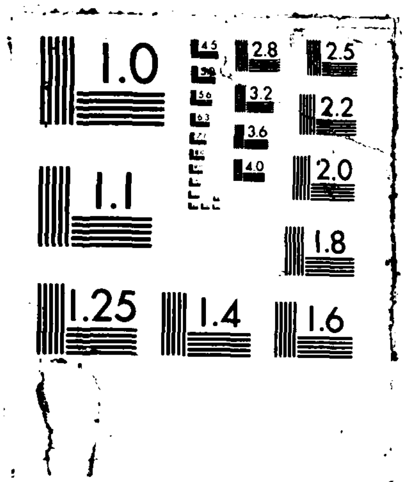
AD-A189 546 NEW YORK TRACON DEMONSTRATION OF PROGRAM RECODING SOFTWARE TRANSLATION AN..(U) FEDERAL AVIATION ADMINISTRATION TECHNICAL CENTER ATLANTIC CIT.. AUG 87 1/2

UNCLASSIFIED DOT/FAA/CT-87/33 DTFA03-85-C-0058 F/G 12/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

AD-A189 546

(2)

DTIC FILE COPY

DOT/FAA/CT-87/33

FAA Technical Center
Atlantic City International Airport
N.J. 08405

New York TRACON Demonstration of Program Recoding Software Translation and Verification Methodology Document

Data Transformation Corporation
8121 Georgia Avenue
Silver Spring, Maryland 20910

August 1987

Final Report

This document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161.

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturer's names appear herein solely because they are considered essential to the object of this report.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/CT-87/33	AD-A184 546	

4. Title and Subtitle
New York TRACON Demonstration of Program Recoding Software Translation and Verification Methodology Document

5. Report Date
August 1987

6. Performing Organization Code
ACT-101

7. Author(s)
Data Transformation Corporation

8. Performing Organization Report No.
DOT/FAA/CT-87/33

9. Performing Organization Name and Address
Federal Aviation Administration
Technical Center
Atlantic City International Airport, N.J. 08405

10. Work Unit No. (TRAIS)

11. Contract or Grant No.
DTFA 03-85-C-0058 (16)

12. Sponsoring Agency Name and Address
Department of Transportation
Federal Aviation Administration
ATC Automation Division
Washington, D.C. 20591

13. Type of Report and Period Covered
Final Report

14. Sponsoring Agency Code
APM-200

15. Supplementary Notes
IBM and Pailen Johnson Associates (PJA) supported the Data Transformation Corp. (DTC) in the performance of this effort.

16. Abstract

This document is the concluding report in a project whose objective was to convert in a reasonably short period of time, machine dependent software to a higher order language capable of running on any general purpose computer. A subset of the New York (N.Y.) TRACON (version A5.04) software was chosen for conversion, specifically, the tracking algorithms. The effort concluded with a demonstration of the converted software running on an IBM 3080 processor and was presented on a Sony display. The present UNIVAC ULTRA programs were converted to ADA/PDL and the system implemented in PASCAL. Over 53,000 lines of ULTRA source code were converted to 47,000 lines of ADA/PDL (including commentary) and then to 32,000 lines of PASCAL (without commentary). The project was concluded in 9 months with the successful demonstration.

17. Key Words
New York TRACON
ARTS IIIA
ADA/PDL

18. Distribution Statement
This document is available to the U.S. public through the National Technical Information Service, Springfield, Va. 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
UNCLASSIFIED	UNCLASSIFIED	124	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

EXECUTIVE SUMMARY

This document is the concluding report in a project whose objective was to convert in a reasonably short period of time, machine dependent software to a higher order language capable of running on any general purpose computer. A subset of the New York (N.Y.) TRACON (version A5.04) software was chosen for conversion, specifically, the tracking algorithms. The effort concluded with a demonstration of the converted software running on an IBM 3080 processor and was presented on a Sony display. The present UNIVAC ULTRA programs were converted to ADA/PDL and the system implemented in PASCAL. Over 53,000 lines of ULTRA source code were converted to 47,000 lines of ADA/PDL (including commentary) and then to 32,000 lines of PASCAL (without commentary). The project was concluded in 9 months with the successful demonstration.

Table of Content

Table of Appendix

Table of Figures

Preface

This report is sponsored by the Federal Aviation Administration (FAA), Technical Center, Atlantic City Airport, New Jersey 08405. Funding for the effort was provided by the FAA, Contract number DTFA 03-85-C-00058 MOD 16, Contract description: New York TRACON demonstration of program recoding.

FAA's sponsorship and management oversight for this contract was provided by APM-220 from the FAA's Washington D.C. office, and by ACT-101, from the Atlantic City Technical Center. This contract was administered as a Cost Plus Fixed Fee (CPFF) contract of nine months duration.

The New York TRACON demonstration of program recoding project was implemented by the team of Data Transformation Corporation (DTC), International Business Machines (IBM), and Pailen Johnson Associates (PJA), under DTC contract 42-37G.

1.0 Summary and Conclusions

The team of DTC, IBM, and PJA completed the recoding of a subset of the current New York TRACON ARTS IIIA software and successfully demonstrated its execution on a System/370, running under MVS/RTX. This activity proved that the tracking function from the existing operational system can be translated into a higher order language and rearchitectured to run in a different system configuration. The project was completed on schedule (in nine months) and within budget.

The team provided a final two hour demonstration on May 29, 1987. The demonstration used an FAA-provided input file, containing the recorded output of a current FAA New York TRACON system run. The contractors and the FAA compared the output recorded from the demonstration run with the New York TRACON generated output and found no unanticipated discrepancies. The analysis verified that the recoded tracking algorithms are functionally equivalent to the original.

The demonstration was run on an IBM 3083 processor, connected to a modern situation display. The tracking outputs, including lists and full data blocks, were presented on the display in real-time, providing visual evidence that the recoded software performed correctly.

The primary objective of the demonstration was to prove that, in a reasonably short period of time, real-time air traffic applications can be transported from one processor and unique instruction architecture to another general architecture without affecting their functional or computational performance. The process included extracting the functional requirements from an existing, proven system that accomplished the primary mission (e.g., the tracker), plus any function that is necessary to support its correct operation (e.g., PSRAP, keyboard message processing, etc.), plus any function that is needed to provide verification of it correct execution (e.g., CDR extraction, display, etc.). Operational data in the form of continuous data recording extractions were used for verification. Computer performance was monitored to determine successful operation of the application functions and the viability of the selected processor and commercial-off-the-shelf operating system.

Several key factors contributed to the project's success:

- The software development laboratory, including all equipment and software tools, was available at the start of the contract.

- The team used a consistent and proven set of development methods and followed the standard software development life cycle; including, requirements analysis and architecture, two levels of design, incremental software builds, design and inspections, and an independent software integration and test team.

- A formal software architecture was developed that insulated the applications and the applications programmers from the system operating environment.

- A complete data element dictionary mapping the existing ULTRA variables to the recoded variables was developed.

- The team's formal design methods, used in conjunction with Ada, led to a design marked by independent modules, with no global database and precisely-defined interfaces. The design was recorded in an Ada process design language (PDL).

- Early in the program, the system inputs were converted to the new format. The input variables were defined as Ada data types to ensure consistency throughout the entire software system. (The CDR Editor, for example, used the same data-types as Retrack, the system input driver.)

- The software was recoded in Pascal/VS, which made the transition from PDL/Ada easy. Pascal/VS is a strong data-typing language with well-defined rules which allow many types of errors to be determined at compile time rather than execution time.

Over 53,000 lines of ULTRA were converted to over 83,000 source lines of Pascal/VS code, including instructions, data, and commentary. (Approximately 62% of the Pascal/VS source lines are comments.) The system rearchitecture and translation of the ULTRA to Pascal was effected by the development of two levels of PDL/Ada. Primarily because of Ada and Pascal/VS, there were no major software errors when the final demonstration was given. Seventy-eight errors were generated and resolved during software integration and testing.

Recoding an existing software system to run on an instruction architecture that is different from the original has significant advantages over an approach where the system is re-specified and re-designed:

- The existing software source code is ultimately the most reliable specification.

- Recoding avoids the errors that would result in developing new engineering requirements and functional specifications.

- Proving source code equivalence is more reliable and more cost-effective than a full-scale verification and validation testing program.

In short, because recoding takes advantage of the profound effects of evolution, it results in a converted system that is more reliable (as it leaves the factory) and far less expensive to develop. The error-prone processes of discovery and invention are minimized.

2.0 Introduction

The information contained in the remainder of this document represents the chronological sequence of activities utilized in the development of the New York TRACON Demonstration of Program Recoding Project (referred to as the Demonstration System). It is subdivided into the following four sections:

- o Translation Methodology
- o Verification Methodology
- o Project Management
- o Statistical Summary

Translation Methodology (Section 3.0)

This section describes the definition and approach to the system analysis, design, code, development, tests, and reviews.

Verification Methodology (Section 4.0)

This section describes the expected behavior of the software, the methods and standards used to measure the reengineered software, the development approach and standards, software traceability, software testing, and reviews.

Project Management (Section 5.0)

This section describes the tools, methods, guidelines, and standards used to efficiently manage this project.

Statistical Summary (Section 6.0)

This section provides a statistical summary of the results of the system's development and implementation, including an analysis of the SLOCs for each development phase (ULTRA, PDL/Ada, Pascal/VS), design issues, and PTRs. Also included in this section is information pertaining to a poll of each individual developers approach to the *reengineering process*, a graph illustrating the results of the questionnaires, and the raw data collected from this interview process.

2.1 Scope

This document describes the methods used to convert a subset of the current New York TRACON ARTS IIIA software from a multi-processor/machine language instruction architecture to a uni-processor/high order language System/370. It illustrates that a machine dependent language like ULTRA, can be reengineered to a High Order Language (HOL) adaptable to a general purpose machine and operating system architecture. Michael J. Lyons and Raymond Jozwik in an article for Government Computer News (GCN) said, "Software reengineering provides a less costly, proven alternative to the time-consuming and risky from-scratch rewrites so many agencies are facing today. Through reengineering, organizations have realized a 20 percent to 40 percent reduction in maintenance costs, extended the useful life of systems and done so for less than half the cost of a start-over approach". Although the information contained in the GCN article applies to sequential data processing programs and not real-time systems, and thus is not directly applicable to this ULTRA reengineering project, it contains information relevant to reengineering efforts in general. This document provides the detail description of each step used in the demonstration system reengineering effort, including the methods, analytical and developmental approach, architecture, design, coding, testing, implementation, performance, and management standards.

2.2 Project Objectives

The objective of the Demonstration System project was to prove the validity of the software translation and verification methodology used to recode a subset of the New York TRACON Operational program. To achieve this, the contractor demonstrated and documented an efficient translation (i.e., reverse engineering) of the existing New York TRACON Operational software ULTRA source code subset into the Process Design Language (PDL)/Ada. The resultant PDL/Ada was used to generate Pascal/VS Higher Order Language (HOL) which was capable of execution on a general purpose, commercial, target computer and operating system. The newly generated PDL/Ada and HOL was functionally identical and directly traceable to the existing New York TRACON version A5.04 ULTRA source code. The contractor provided means to allow the FAA to verify the recoding (reverse engineering) of the source code.

The translation methodology provides the procedures and tools used to generate an accurate system translation. The verification methodology procedures and tools are used for the verification of the translation process.

This effort was not intended to convert the entire N.Y. TRACON ARTS IIIA software, or to change and/or improve the existing algorithms. Display evaluation or development, performance measurements, and detailed computer sizing, were not objectives of this contract.

2.3 Applicable Documents

It is recommended that a copy of the associated Demonstration System's glossary be available when reading this document, because some of the terms used are unique to the Demonstration System's applications.

2.3.1 Project Documentation

- o A001 Program Management Plan
- o A002 Requirements Analysis Document
- o A003 Top Level Design Document
- o A004 PDL and Traceability Matrices
- o A005 Test Plan
- o A008 Program Listings
- o A010 PDL Reference Manual

2.3.2 FAA Documentation

- o N.Y. TRACON A5.04 Program Listings, and Continuous Data Recording (CDR) files on magnetic tapes (GFE)
- o N.Y. TRACON A5.04 Coding Specifications (GFE)
- o N.Y. TRACON Computer Program Functional Specification (GFE)
- o N.Y. TRACON A5.04 CULL Listings (GFE)
- o N.Y. TRACON Supplement To ARTS IIIA (System Design Data)

3.0 Translation Methodology

A basic goal of the project was to adhere to modern software engineering principles such as data ownership, data hiding and package definitions. The project team used a conversion methodology based on the standard software system development life cycle. The major life cycle steps are:

- Requirements Analysis
- Design
- Implementation
- Software Integration and Testing (SWIT)
- Demonstration

The conversion methods considered the selection of the hardware configuration, operating system, and high order language.

The demonstration system architecture mirrors the New York TRACON A5.04 system architecture by structural components only, not in the content of these components. The demonstration system consists of two subsystems: 1) the real-time operational subsystem and 2) the offline support subsystem. The capabilities of the demonstration system's hardware (processors and peripherals) were mapped to its counterpart, the New York TRACON A5.04 system hardware and its interfaces, to ensure that all functionality was provided.

The demonstration system was constructed from release A5.04, revision H of the New York TRACON operational system. The critical components and services provided by the ARTS IIIA system were mapped by the architectural committee and defined as the minimal requirements to be satisfied by the demonstration system (See Figure 3 "Mapping of MPE Services to MVS/RTX" on page 20). This approach ensured that the recoded system was functionally equivalent to the current operational system. It also made the application work much simpler by helping to identify interrelationships and interfaces, and this relieved the developers from real-time and MVS/RTX considerations and allowed them to concentrate on their specific application areas. From the architectural committee's baselines and analysis, the minimal acceptable goals, objectives, and standards, and the operating system's behavioral criterion were defined. They include:

- Automatic scheduling and dispatching of tasks

- A system architecture that was processor-independent and would accommodate the addition of new software components.

- An architecture that would accommodate a distributed hardware implementation.

- Replacing the encoded lattices with static tables. The encoded lattices defined rules for concurrent and sequential execution of ARTS applications among the various members of the IOP system. The static tables define the number of tasks, the priority assigned to each application task and the resources required for each. Lattices, which are inherent in a multi-processing architecture, are not required in a uniprocessing environment.

- Eliminating the need or constraints of maximizing resources, because tasks wait for work and execute without these constraints in the demonstration system.

- Use of a commercial general purpose operating system (MVS), instead of special purpose, machine and dependent monitors (MPE, NAS Monitor).

- Providing alternate layers of task, storage, timing and recovery controls.

- Reducing the operating system overhead through the use of the Real-time Executive (RTX).

- Shielding the application software from the operating system control program services by a layer of application services. The application services initialize and terminate the tasks, supply timer services to the tasks, monitor the execution time of each task, and handle all inter-task communication.

- Isolating the algorithms from applications.

- Maximizing the source code traceability.

- Ensuring task ownership of necessary data.

- Assisting task communication through the use of well-defined messages (similar to processors in a network).

- Localizing the I/O requests (interface with the input CDR file, output CDR file and the display hardware.)

- Aiding the precise definition of global data.

- Helping tasks provide internal queueing to ensure consistency.

- Localizing application requests for service in programs (called gateways) which interface with RTX to send and receive work, time or schedule events, and acquire resources (global data) outside its boundaries.

The demonstration system architecture was developed within the following boundaries and limitations:

1) Recoding was limited to a subset of the functions of the N.Y. TRACON A5.04 system. This subset included the basic tracking functions, front end (PSRAP), and man-machine interface processing.

2) The demonstration system does not support error recovery.

3) The demonstration system does not process bulk flight data.

4) The demonstration system does not provide the capability to interrupt or allow operator input during system operation.

5) The demonstration system maintains the identical overall data flow.

6) The demonstration system maintains the algorithmic processing of the system work.

7) The demonstration system will provide no interface with ARTCCs; it only accepts interfacility data from a CDR tape.

8) The demonstration system provides interfaces only with disk files and a single situation display.

9) The demonstration system processes simulated input from DEDS keyboards, magnetic tapes, and from interfacility interfaces through the Retrack CDR records written to a file resident on a 3380 Direct Access Storage Device (DASD).

3.1 Requirements Analysis

Requirements Analysis consisted of the following sub-phases:

a. Understanding the task to be performed.

The following steps were taken in this effort:

(1) A high level understanding of the application to be re-engineered was gained. This included an understanding of the application (air traffic control), the operating environment and tools, the hardware environment, and the real world interfaces (inputs and outputs to the application).

(2) Identifying and interpreting the contract requirements. The SOW and FAA directions were used as requirements.

The following requirements were found in the SOW :

(a) Software generated by the conversion effort must be functionally equivalent and directly traceable to the NY TRACON system.

(b) A HOL must be used as the target language.

(c) A PDL (Process Design Language) must be used to record the design.

(d) The CDR Editor shall have the same functional capabilities as the NY TRACON CDR Editor for version A5.04 and produce hardcopy identical in content and format.

(e) RETRACK must control the timing of the sensor inputs.

(f) Priorities for software translation were assigned.

(g) Use of existing off the shelf software components should be maximized.

Additional assumptions based on the SOW were:

(a) Commercially available hardware would be used.

(b) A Commercial Off-The-Shelf (COTS) operating system would be used.

(c) Modern software engineering principles would be adhered to.

(3) Understanding the system (hardware or software) architectural differences.

This was accomplished by:

(a) Analyzing the MPE services and how they could be supplied by MVS/RTX.

(b) Architecturing the system to use multi-programming on a uniprocessor to emulate the multi-processing environment of the existing system.

b. Developing an architecture for the re-engineered system. A summary of the steps taken can be found below and details can be found in section 3.1.1 of this document.

(1) High-level blocks were defined to represent the functional components of the re-engineered system (PSRAP, KEYBOARD, etc.)

(2) The functional components and their dependencies on other components were analyzed to determine the work flow through the system.

(3) The architecture was formally recorded and reviewed.

c. The requirements for data use and access were analyzed. A summary of the steps taken can be found below and details can be found in section 3.1.2.

(1) The subset of data necessary for the recoded system was determined.

(2) The applications that used each particular piece of data were identified.

(3) Ownership of the data was assigned to a specific application.

d. The NAS-MDs, the coding specifications and the source listings from the NY TRACON system were analyzed to determine the sub-functions that would be implemented to meet the SOW. For details on this process, refer to 3.1.2 of this document.

e. Work products were created and reviewed; after they were determined to be acceptable, they were used as input to the design phase.

The work products that were delivered to the FAA in the Requirements Analysis document are:

(1) a formally recorded architecture,

(2) the functions to be recoded (by NAS-MD)

(3) and a data element dictionary.

3.1.1 Architectural Analysis

During the requirements analysis step, the architecture for the recoded system was derived and formally recorded. The architecture was recorded in two parts: (1) the rationale for selecting the architecture and (2) the definitions and rules of expected behavior of the operational software.

The architectural considerations included the system architecture and the software architecture. The system architecture includes the functionality of the current New York TRACON A5.04 system hardware configuration, the mapping of software to hardware, and the flow of control and data through the system. The software architecture consists of a description of the operating environment, its development and operational (real-time) subsystems, mapping of the system application tasks and system operations tasks to the software, units of software, functions, data bases, allocations, interfaces, synchronization and resource use.

The primary architectural requirement was that the demonstration system algorithms perform functionally the same as the A5.04 TRACON system but execute on a uniprocessor under a COTS operating system.

In a TRACON operational system, target reports are received from the external world, processed by the applications and the results are presented to the users of the system. This flow of data suggests a "pipeline" through the system and is reflected in the architecture.

Figure 1 depicts the architecture that was defined. It is an implementation independent architecture that could be implemented in a uniprocessor or multi-processor environment.

Since the target operating system was MVS/RTX, the services supplied by the MPE were reviewed and mapped to services that were available in MVS/ RTX. The mapping is illustrated in Figures 2 and 3.

As the architecture evolves, a need was identified for a set of software to support the architecture and its implementation on specific hardware and to supply operating system services to the applications. The following applications were defined:

a. initialization and termination

b. the message passing

c. timer services

d. and input and output services (DEDs).

The package concept was used to record the architecture and to satisfy the requirements for data ownership and encapsulation.

During the requirements analysis step, the application packages were defined and the flow of work through the system was characterized. Work flow diagrams were developed and analyzed. The behavior and rules for each category of tasks were defined and recorded. Functions were restructured to run under the Multiple Virtual Storage (MVS) operating system to accomplish the

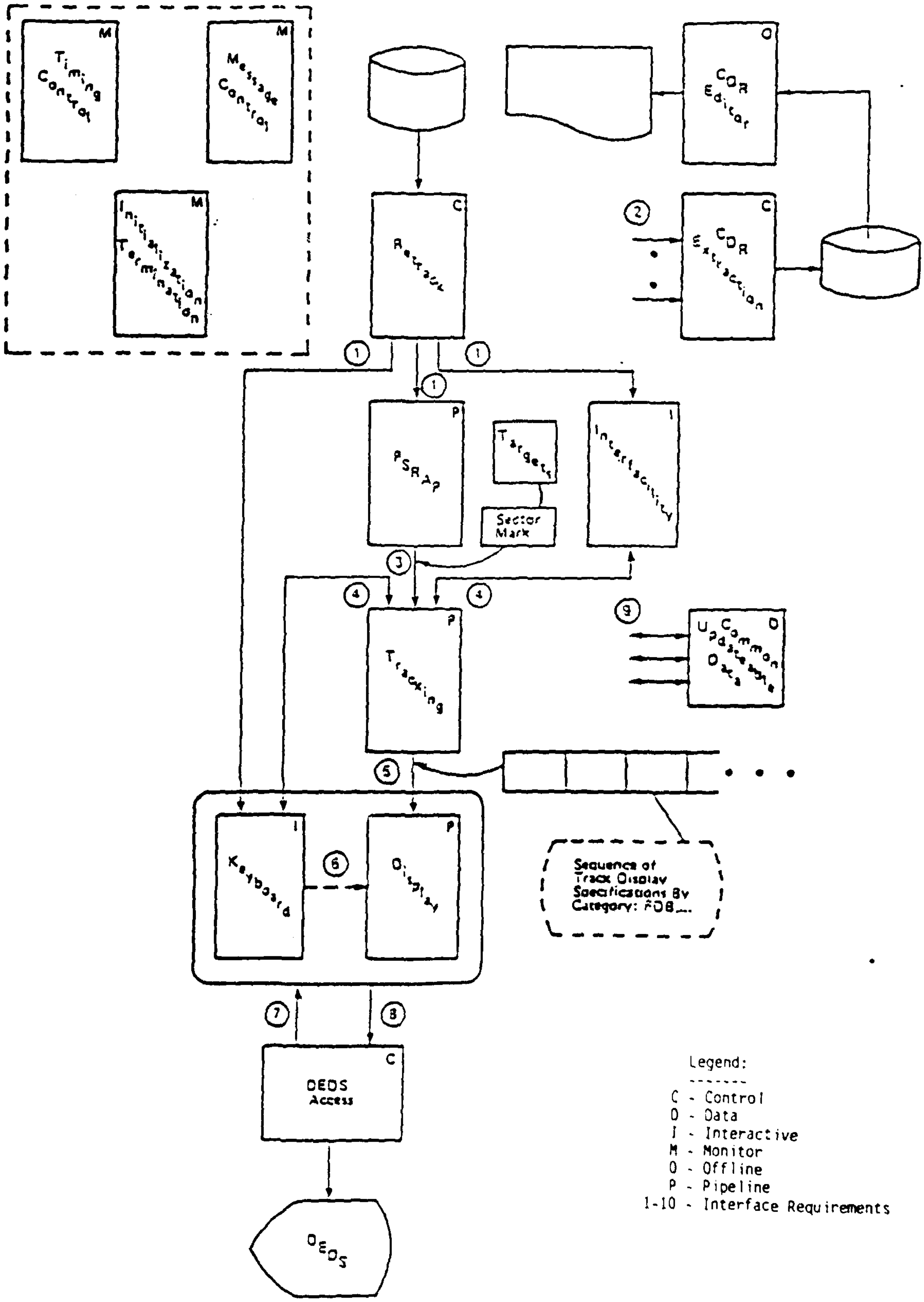


Figure 1. N.Y. TRACON Demonstration Operational Software Architecture

SUMMARY OF IOP MPE SERVICES (ESRs)	
Service Name	Description of Service
1. EXIT	1. Exit from a task
2. CHAIN/BUFFER	2. Perform I/O to peripherals that do not have separate handlers
3. REQUEST PERIPHERAL ASSIGNMENT	3. Request channel and IOP number for specified peripheral
4. INTERCEPT INTERRUPT	4. Request device interupts be routed to application
5. TERMINATE I/O	5. Stop I-O for specified peripheral
6. ENABLE/DISABLE CHANNEL	6. Enable or disable channel interrupts
7. REQUEST ALTERNATE CHANNEL	7. Switch channel for peripheral
8. REQUEST ALTERNATE PERIPHERAL	8. Switch between primary and backup peripheral
9. SEL. LATTICE IDX.	9. Alter starting point of next lattice
10. REQ. POPUP IDX.	10. Find task index of popup task
11. SCHEDULE POPUP	11. Dynamically schedule/deschedule popup
12. SCHEDULE PERIODIC POPUP	12. Dynamically schedule/deschedule popup task
13. CYCLE CONTROL	13. Modify cycle advance time
14. INSERT LATTICE	14. Insert new lattice as next lattice
15. ABORT LATTICE	15. Stop current lattice and start next one
16. INITIATE OTHER	16. Allow execution of next lattice
17. DEBUG SNAPSHOT DUMP	17. Dump debug data on TTY or printer
18. DECL. CRIT. DATA	18. Define data to be recorded
19. RCD. CRIT. DATA	19. Record all critical data
20. LOAD CRIT. DATA	20. Load all critical data
21. DISC	21. Schedule DISC popup task
22. IMT	22. Schedule tape popup task
23. MSP	23. Add message to printer queue, schedule printer popup task
24. TTY	24. Add message to TTY
25. SCATTER INTERRUPT	25. Perform global interrupt
26. ON-CALL LOAD	26. Load and start an on-call program (disk)
27. BACK-UP PGM LOAD	27. Spec'y program load under degraded conf.
28. CAPTURE CMC INTERRUPT	28. Establish user interrupt handling on CMC
29. REQUEST ALTERNATE CMC PERIPHERAL	29. Switch between CMC subchannels
30. CMC I/O	30. Initiate I/O on specific ESI channel
31. SWITCH TTY	31. Switch messages between TTY and printer

Figure 2. MPE SERVICES

MAPPING OF IOP MPE SERVICES (ESRs) PART 1
TO MVS and RTX

MVS/RTX FUNCTION	MVS/RTX* SERVICE	ESR #	REPLACED MPE ESR
* Services beginning with 'G' are MVS/RTX Macros. Other entries are MVS standard system services.			
WORK MANAGEMENT			
Task Creation, Control and Scheduling	GQRETUR	1	Exit ESR
	GQWORK	9	Select Lattice Index
	GQWORK	10	Request Popup Index
	GQWORK	11	Schedule Popup
	GQWORK	12	Schedule Periodic Pop.
	GKWORK	13	Cycle Control
	GQWORK	14	Insert Lattice
	GQWORK	15	Abort Lattice
	GQWORK	16	Initiate Other
DEVICE ACCESS METHODS			
3274 Communications Controllers I/O	VTAM	24	TTY ESR
	VTAM	28	Capture CMC Interrupt
	VTAM	29	Request Alternate CMC Peripheral
	VTAM	30	CMC I/O
3480 Tape Access Method	READ/ WRITE	22	IMT ESR
Online Print Interface	GKSPRINT	23	MSP ESR
Data Management Service	GKREAD	21	DISC ESR
	GKWRITE	21	DISC ERR
SERVICE LEVEL I/O			
	IOS	2	Chain/Buffer Request
	IOS	3	Request Peripheral Assignment
	IOS	4	Intercept Interrupt
	IOS	5	Terminate I/O
	IOS	6	Enable/Disable Channel
	IOS	7	Request Alternate Channel
	IOS	8	Request Alternate Peripheral
	IOS	31	Switch TTY ESR

Figure 3. Mapping MPE SERVICES to MVS/RTX

equivalent task organization and timing that was accomplished by the lattice architecture in the current multi-processing environment with the Multi-processor Executive (MPE).

The project's team used Pascal/VS HOL for most of the translation. IBM Assembler H was used to construct interfaces between RTX system services and the applications. Specific examples of this include:

- o Application bridges to create and preserve state data vectors and to monitor task elapsed time.
- o Services to obtain system time of day.
- o SEND/RECEIVE interface to RTX to provide application tasks the ability to enqueue and dequeue work among other application tasks.

3.1.2 Requirements Analysis of the TRACON NAS-MDs

During the requirements analysis phase, the contractor was required to perform a comprehensive analysis of the NY TRACON software, represented by the GFE NY TRACON software listings and CPFSs. The portion of the requirements analysis described in this section was the detailed analysis of the software requirements, based on the ARTS IIIA Computer Program Functional Specifications (CPFS) for version A5.04. This work was organized by NAS-MD. In addition to the CPFS, the Statement of Work required that the recoding (translation, text, demonstration, and verification) be accomplished in accordance with the schedule and priority scheme identified in Section 3.3.3, "Operational and Support Software Component Priority" within the New York TRACON Demonstration of Program Recoding: Statement of Work. The support software was to include a CDR Reduction Program and a driver (Retrack type). There were three priorities identified in the statement of work. Ingeneral, priority 1 items were coded, priority 2 items were optionally coded, and priority 3 items were not coded. Further information is provided on exceptions to this guideline.

Each section contains an introductory paragraph, the analysis by NAS MD subsection, and a discussion of additional capabilities, if there are any. If a subsection contains a functional capability that is being converted from ULTRA to Pascal/VS, it is identified under the heading "Recoded" with a "Yes"; if the function is not a software function, or is being replaced by commercial software, or is not being considered for the demonstration, or contains administrative information only, and so on, it is identified under the "Recoded" heading with a "No.". In either case, the rationale is included.

The template shown in Figure 4 was used to record our analysis. For the most part, the functional analysis proceeded based on priority: Priority 1 items were all coded, Priority 2 items were omitted, except for Display Output, and Priority 3 items were omitted.

The following exceptions arose from this:

- Interfacility

  Though interfacility was a priority 3 function, there was need in the recoded system to provide for a buildup of flight plan data. By processing FP (and DA messages for the FP), AM, and CX messages from the CDR file, a flight plan data base could be created for association with tracking data. No additional interfacility messages were processed; no hardware interface was implemented.

- Keyboard Input Processing

  Our analysis indicated that the extraction of keyboard data was after KIP had processed. Therefore, it was not necessary to record KIP, even though it was a Priority 1 item.

Other issues that were addressed were:

- CDR Conversion

  The recoding was performed using the PASCAL language. It does not allow reference to the assembler bit-specific formats used in the current system. Therefore, the CDR file had to be converted to PASCAL format by a combination of assembler and PASCAL code (after being duplicated from 7-track tape format to the 9-track tape format). We decided to perform the conversion offline for the following reasons:

  - The entire process does not have to be repeated on every demonstration run

  - The operational system does not have to be concerned with the formats in the current ARTS IIIA system

  - The Retrack input formats are identical to the CDR Extractor output formats, allowing for the possibility of:

    - Running the CDR Editor against the converted GFE tapes (which we did to test the Editor and obtain multiple copies of the GFE output)

Sub-section	Title	Recoded
x.x.x.x.x.x.x	YYYY YYY YYYYYYY YY	Yes
	For any part or for the entire sub-section, enter the rationale for recoding, such as function is a priority 1 item, or enter the rationale for not recoding: choose one of the following or add rationales as needed: This is a priority 1 function ... (include a reason if we are not doing it). This is a priority 2 function ... (include a reason if we are not doing it). This function is derived from the FAA requirements and is required to maintain an integral system. This section is administrative and contains no demonstrable functions. This section provides technical content but contains no demonstrable functions. Refer to Section x.x.x of this document for the rationale. An equivalent function is being provided by (and identify the products or capabilities we are substituting.) (include explanations that will clarify system issues, such as our use of a KVDT instead of their CDT.) This function is not required by the FAA and is not required to maintain an integral system.	

Figure 4. Requirements Analysis Template

ooo Using the CDR Extractor file as input to Retrack

The requirements analysis for the CDR Conversion program identified the messages that needed to be converted; input and output messages not processed by the demonstration system were not converted.

oo CDR Extraction

In the recoded system, CDR Extraction would not have global access to the data that is extracted in the current NY TRACON system. Therefore, the tasks that generated the data now had to send the data to the extraction task. The extraction task buffered the data onto the CDR file.

oo Retrack

Retrack in the current NY TRACON system has the capability to search the operational Central Track Store (CTS) and other operational data that it needs. In the recoded system, tasks, including Retrack, are not able to access data owned by other tasks (CTS was owned by Tracking). Therefore, handling of input messages was different. Retrack had to become more of a driver, sending each CDR message to the input queue of the program that would process the message.

The following items were added to Retrack in the demonstration system:

ooo Discarding Target Report and Radar Only Target messages until the first sector mark for that sensor was received. Blocking of Target Report and Radar Only Target messages for PSRAP.

This minimized the number of Sends from Retrack to PSRAP.

ooo Matching of NY TRACON generated DA messages to FP messages sent by the ARTCC. Saving the ACID based on the TCID from the CDR input file.

This enabled the demonstration system to process only those FP messages (and subsequent AM and CX messages) that were accepted by the NY TRACON system.

ooo Fabrication of Flight Data Entry controller messages from Tracking Data CDR messages. This enabled the demonstration system to build a flight data base complementary to the interfacility data base to allow for association with target reports.

oo Display

The display output processing had to support the situation display that was used in the demonstration.

3.1.3 Data Base Analysis

As part of the requirements analysis, the project team generated two data element dictionaries (DED): 1) an operational system data element dictionary, and 2) a system and site adaptation parameter data element dictionary. The data dictionaries identified for each member in the data base, the current name, the new Pascal name, the procedures that referenced it, and how each were defined and used.

To accomplish this the project team utilized the FAA GFE (ULTRA listings of the source programs) and the A5.04 CULL listings. This effort enabled the developers to gain a concise understanding of each element, how and where it was used, identify dependent routines, local, and global parameters, and understand the content and complexity of the element.

The analysis of the operational system data element dicticnary was completed during the requirements analysis phase. All the developers reviewed their code to understand if the element was set or used in the program. When this work was completed, the Operational and Support Software Component Priority (Section 3.3.3 of the Statement of Work) was used as a basis to determine whether the element was needed in the demonstration system. An example of the operational parameter data element dictionary is shown in Figure 5. See Appendix B-1 for a description of the layout of the data element dictionary.

The system and site adaptation parameter data element dictionary was developed in two separate components: elements referenced by the system data base (e.g., SDB1, SDB1RO, SDB2, DBASEC, DBASED, and DBASEE) located in the mapping tables shown in Figure 6; and the other elements defined by the site adaptation procedures and not referenced by the system data base (e.g., CSITEQ, DSITEQ, MSITEQ, TSITEQ, SYSEQ0, and TI). Figure 7 contains parameters relevant to system and site data and program controls. A description of the layout of the mapping tables is located in Appendix B-2.

The system and site adaptation parameters differs from the operational system data element dictionary by the exclusion of the columns depicting which procedures set or define the parameter. The elements defined in the system and site adaptation parameter data element dictionary were mainly constants and array definitions. The system and site parameters

Record#	COMPANY	DATABASE	DATABASE	PAGENUM	P1	S1	P2	S2	P3	S3	P4	S4	P5	S5	P6	S6	P7	S7	P8	S8	P9	S9	P10	S10	TYPE	VARNAME	DBNAME	NEWPGNUM
1		APACK	SDB2	37	KOF	0	MPEP	0		0		0		0		0		0		0		0		0	C	disp_pack_store_disp	DISP	0
2		AQLR2T	SDB2	7	COMC	2	TRAD	2		0		0		0		0		0		0		0		0	C	key_two_encode_disp	DISP	0
3		AQLR3T	SDB2	7	COMC	2	TRAD	2		0		0		0		0		0		0		0		0	C	key_three_encode_disp	DISP	0
4		AQLRCT	SDB2	7	COMC	2	TRAD	2		0		0		0		0		0		0		0		0	C	key_one_encode_disp	DISP	0
5		AWDT	SDB2	39		0	COMC	2	TRAD	0		0		0		0		0		0		0		0	A	aword_table	DISP	0
6		CRTYXT1	SDB2	5	AUT	2	IFO	0	KIP	0	RKIP	0		0		0		0		0		0		0	I	xcoord_range	DISP	0
7		CRTYXT2	SDB2	5	AUT	2	IFO	0	KIP	0	RKIP	0		0		0		0		0		0		0	I	ycoord_range	DISP	0
8		DBCOUNT1	SDB2	40	TRAD	3	COMC	3	DOP	0	MPEB	0		0		0		0		0		0		0	I	db_alt_counter	DISP	0
9		DBCOUNT2	SDB2	40	TRAD	3	COMC	3	DOP	0	MPEB	0		0		0		0		0		0		0	I	ss_counter	DISP	0
10		DBCOUNT3	SDB2	40	TRAD	0		0		0	MPEB	0		0		0		0		0		0		0	I	fdb_counter	DISP	0
11		DCONT	SDB2	3	KOF	0		0	MPEB	0	MTGA	0	MTGCT	0	PDOP	3	QLOOK	0	RCOMP	0	RDOP	0	RKIP	0	A	para_tabl_disp	DISP	0
12		DCONT	SDB2	3	AUT	2	COMA	2	COMB	3	COMC	3	CRIT	0	DOP	0	IFI	0	IFO	0	KOF	2	KIPM	0	A	param_tabl_disp	DISP	0
13		DCONT	SDB2	3	RTDOP	0	SWABS	0	TDOP	3	TRAD	3		0		0		0		0		0		0	A	parm_tabl_disp	DISP	0
14		DFLAGT	SDB2	4	AUT	2	COMC	2	CRIT	0	DOP	0	IFO	0	KIP	0	KIPM	0	KOF	0		0		0	A	param_tab2_disp	DISP	0
15		DFLAGT	SDB2	4	-MPEB	0	MTGA	0	MTGCT	0	PDOP	2	QLOOK	0	RCOMR	0	RKIP	0	RTDOP	0	SWABS	0	TDOP	2	A	parm_tab2_disp	DISP	0
16		DFLAGT	SDB2	4	TRAD	3	KOF	2		0		0		0		0		0		0		0		0	A	param_tab2_disp	DISP	0
17		DSLIMT	SDB2	52	TRAD	2		0		0		0		0		0		0		0		0		0	I	dslimt_trad	DISP	0
18		EMRFT	SDB2	24		0	COMB	1	PDOP	0	RDBMD	0		0		0		0		0		0		0	I	es_rf_counter	DISP	0
19		HJT	SDB2	24		0	COMB	1	PDOP	0	RDBMD	0		0		0		0		0		0		0	I	hj_sa_counters	DISP	0
20		IDUBT	SDB2	15	KIP	0	TROUT	0	COMA	3	TPUR	0	KOF	3		3	TRAD	2	SLINK	0	TPSEC	0	TINIT	0		idupbt_key_trk_info	DISP	0
21		IDOPT	SDB2	52	TDOP	3		0		0		0		0		0		0		0		0		0	I	dis_count_disp	DISP	0
22		TKRCXT	SDB2	6		0	COR	2	COMC	3	CRIT	0	IFD	0	KIF	0	MPED	0	ALOOK	0	AKIP	0	TRAD	3	I	tkrext_misc_disp	DISP	0
23		UAFLT	SDB2	5		0		0	KOF	0	TRAD	2		0		0		0		0		0		0	I	alt_filter_display_up_low_limits	DISP	0
24		CHGFLG	TMP	5	COMC	3	TMP	0	TRAD	3		0		0		0		0		0		0		0	I	chgflg_temp_trk	DISP	0
25		CLDTRK	TMP	5	TMP	0	TRAD	3		0		0		0		0		0		0		0		0	I	cldtrk_temp_trk	DISP	0
26		DSPCLR	TMP	5	TMP	0	TRAD	3		0		0		0		0		0		0		0		0	I	dspclr_temp_trk	DISP	0
27		DTCHGFLG	TMP	6	TMP	0	TRAD	3		0		0		0		0		0		0		0		0	L	alt_chg_flag_trk	DISP	0
28		LKDTRK	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	kdtrk_temp_trk	DISP	0
29		TEMTR1	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	temtr1_temp	DISP	0
30		TEMTR2	TMP	5		0	COMC	3	TRAD	3		0		0		0		0		0		0		0	I	temtr2_temp_trk	DISP	0
31		TEMTR3	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	temtr3_temp_trk	DISP	0
32		TEMTRK	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	temtrk_temp	DISP	0
33		TRADB3	TMP	6		0	COMC	3	TRAD	3		0		0		0		0		0		0		0	I	tradb3_temp	DISP	0
34		TRADB5	TMP	6		0		0	TRAD	0		0		0		0		0		0		0		0	I	tradb5_temp	DISP	0
35		TRKALT	TMP	5		0	COMC	3	TRAD	3		0		0		0		0		0		0		0	I	trkalt_temp	DISP	0
36		TRKCLR	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	trkclr_temp	DISP	0
37		TRKFAM	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	trkfam_temp_trk	DISP	0
38		TRKLCA	TMP	5		0	TRAD	3		0		0		0		0		0		0		0		0	I	trklca_temp_trk	DISP	0

Figure 5. Operational Parameter Data Element Dictionary

****
* DATA ELEMENT DICTIONARY *
* OF *
* SITE & SYSTEM PARAMETERS *
****

COMPANY	DATANAME	DATABASE	PAGENUMBER	P1	P2	P3	P4	P5	TYPE	VARNAME	DBNAME	NEWPGN
DTC	AADTQ	TI	2.0-36	MSITEQ					1	DEDS_PROC_SSFDB_AWORD_DISP_TIME	DISP	
DTC	APBTQ	TI							1	TIME_SHARE_DEAD_TIME_ABD_PARM	DISP	
DTC	ACNTQ	TSITEQ		TINIT					1	NO_SCAN_TARGET_HITS	SPARM	
DTC	ACPZONEQ	TI	2.0-28						1	MISC_TRK_ALLOC_ACPZONE	TRACK	
DTC	AFIRMQ	TI	2.0-26	TINIT					1	MISC_TRK_ALLOC_FIRM_AUTO_AC	TRACK	
DTC	AIHOQ	TI	2.0-4	IDAT					1	IFY_ALLOC_MILES	IFY	
DTC	ANCRESQ	MSITEQ	3.0-34	BRATS	CTIP	DTOD	FPDU	LIMD	1	LENGTH_TTY_INPUT_BUFFER	SPARM	
DTC	ANCRESQ	MSITEQ	3.0-34	MSITEQ	MTGA	MTGCT	SCDU	TSO	1		SPARM	
DTC	ANCRESQ	MSITEQ	3.0-34	MTGA	MTGCT	SCDU	TSO	TI	1		SPARM	
DTC	ANCTK1Q	TI	2.0-28						1	ON_CALL_PROG_INITIALIZE	SPARM	
DTC	ANCTK2Q	TI	2.0-28						1	ON_CALL_PROG_EXECUTE	SPARM	
DTC	ANCTK3Q	TI	2.0-28	BRATS	CTIP	DTOD	FPDU	LIMD	1	ON_CALL_PROG_TERM_ENABL_FLAG	SPARM	
DTC	ANCTK3Q	TI	2.0-28	SCDU	TSO				1		SPARM	
DTC	ANCTK4Q	TI	2.0-28	BRATS	DTOD	FPDU	LIMD	SCDU	1	ON_CALL_PROG_MSG_PREFIX_AREA	SPARM	
DTC	ANCTK4Q	TI	2.0-28	TSO							SPARM	
DTC	ANCTK5Q	TI	2.0-28	CTIP					1	ON_CALL_PROG_INPUT_TTY_MSG	SPARM	
DTC	ANCTK6Q	TI	2.0-28	DTOD	FPDU	LIMD	MTGA	MTGCT	1	ON_CALL_PROG_UNPACK_TTY_MSG	SPARM	
DTC	ANCTK6Q	TI	2.0-28	SCDU	TSO				1		SPARM	
DTC	ANCTK7Q	TI	2.0-29	BRATS	CTIP	FPDU	IFO	KIP	1	ON_CALL_PROG_PASSED_DATA	SPARM	
DTC	ANCTK7Q	TI	2.0-29	LIMD	RKIP	SCDU	TSO		1		SPARM	
DTC	AOFSTLQ	MSITEQ	3.0-14	SDB2					1	AUTO_OFFSET_ENABLE	SPARM	
DTC	ARCSTQ	DSITEQ		IDAT					A	ARTCC_SOURCE_ID	SPARM	
DTC	AREATQ	TI		ALTRKR	ALTRKR1				1	. IT_ALT_ACCEL_REASON	TRACK	
DTC	AREAUQ	TI		ALTRKR	ALTRKR1				1	SEC_ALT_ACCEL_REASON	TRACK	
DTC	ARTSIDQ	DSITEQ		IDAT					A	EBCDIC_SOURCE_ID	IFY	
DTC	ARTSIQ	DSITEQ		IDAT	SDB2				A	NYTRACON_SOURCE_ID	SPARM	
DTC	ARTS1Q	DSITEQ		IDAT					A	NYT_SOURCE_ID_1	SPARM	
DTC	ARTS2Q	DSITEQ		IDAT					A	NYT_SOURCE_ID_2	SPARM	
DTC	ARTS3Q	DSITEQ		IDAT					A	NYT_SOURCE_ID_3	SPARM	
DTC	ARTS4Q	DSITEQ		IDAT					A	NYT_SOURCE_ID_4	SPARM	
DTC	ARTS5Q	DSITEQ		IDAT					A	NYT_SOURCE_ID_5	SPARM	
DTC	ASQ	TI	3.0-16	COMA					1	MINS_PRE_ETA_FP_CHNG_STOR_DISPL	IFY	
DTC	ASR7Q	TSITEQ							1	ASR_BEACON_SUBSYS	SPARM	
DTC	ADR7SQ	TSITEQ							1	ASR_ASSOC_DISPLAY_NO	SPARM	
DTC	ATRNGQ	MSITEQ	3.0-39	MTGA	MTGCT				1	ADAPTED_TRAIN_DISP_FLAG	SPARM	
DTC	ATSQ	TI	3.0-16	COMA					1	MINS_PRE_FIX_ARIV_OVFL_TMPTOCTS	IFY	
DTC	AUTO	SYSEQ0	3.5.2-2	AUT	KOFC	SDB2	SYSEQ1	SYSEQ2	1	AUTO_OFFSET	SPARM	
DTC	AUTTIMQ	IT		AUT					1	TIME_ALLOC_DELTA_AUTO_OFFSET	DISP	
DTC	AVGDTQ	DSITEQ							1	AVG_DISP_DEAD_TIME	DISP	
DTC	AWDQ	TI	2.0-21	KOFA	KOFB	KOFC	MTGA	MTGCT	1	MEM_ALLOC_BUFF_TBPRET_AB_WORDS	KBD	
DTC	AWDQ	TI	2.0-21	SDB2							KBD	
DTC	BCNTQ	TSITEQ		KOFA	KOFC				1	NO_SCAN_DISP_BEACON	SPARM	
DTC	BINBIASQ	TI	2.0-26						1	MISC_TRK_ALLOC_BIN_BIAS	TRACK	
DTC	BLKS	SYSEQ0	3.5.2-2	CRIT	KOFC	MTP	SDB2	SYSEQ1	1	BULK_STORE_FP	SPARM	

Figure 6. System and Site Parameter Data Element Dictionary

```
*****************************
*     MAPPING TABLE FOR     *
* DATA ELEMENT DICTIONARY   *
*            AND            *
*      SYSTEM DATABASE      *
*****************************
```

DATANAME	SECTION	PAGENUM	DATABASE	SDB1	SDB1RO	SDB2	CFGT	CDRD	SUBS	MBUF
AkQ	3.85	3.0-31	DSITEQ			KBDT				
AkQ	3.85	3.0-31	DSITEQ			TBPRET				
AkQ	3.85	3.0-31	DSITEQ			SYMT				
AAijQ	3.55.14	3.0-24								
AADTQ	2.158	2.0-36								
AAFijQ	3.55.15	3.0-24								
AAZaiQ	3.33	3.0-9	TSITEQ		AAFXP					
AAZmiQ	3.30	3.0-8	TSITEQ	OVAR2						
AZZmiQ	3.31	3.0-8	TSITEQ	OVAR1						
AAZLmiQ	3.31	3.0-8	TSITEQ	OA2OR2						
AAZLmiQ	3.31	3.0-8	TSITEQ	OA3OR3						
AAZLmiQ	3.31	3.0-8	TSITEQ	OA1OR1						
ABEAT	2.165	2.0-37								
ABIASQ	2.26	2.0-5								
ACMQ	8.3.1	8.0-9								
ACNTQ	3.54.13	3.0-17								
ACPZONEQ	1.126	2.0-28								
ACQ	8.4.1	8.0-15								
ACTYPT	2.165	2.0-39								
ADijQ	3.55.14	3.0-24								
ADAQ	8.2.1	8.0-1								
ADBNQ	3.119	3.0-36	MSITEQ			DSLIMT				
ADDBP	2.41	2.0-8								
AFIRMQ	2.126	2.0-26								
AFIXAiQ	3.32	3.0-9	TSITEQ	AFXP	AAFXP					
AFRMQ	2.27	2.0-5								
AIFRQ	3.20	3.0-5	DSITEQ			VIALT				
AIHOQ	2.17	2.0-4								
ALALT	7.5.3	7.0-44								
ALARMQ	7.6.2	7.0-45								
ALT	2.165	2.0-38								
ALTiQ	2.2	2.0-1								
ALTMASKQ			MSITEQ			ALTMASK				
ANCRESQ	2.127	2.0-29								
ANCRESQ	3.109	3.0-34								
ANCTK1Q	2.127	2.0-28								
ANCTK2Q	2.127	2.0-28								
ANCTK3Q	2.127	2.0-28								
ANCTK4Q	2.127	2.0-28								
ANCTK5Q	2.127	2.0-28								
ANCTK6Q	2.127	2.0-28								
ANCTK7Q	2.127	2.0-29								
AOFSTLQ	3.46	3.0-14								
APAijQ	3.55.9	3.0-22a								
APALijQ	3.55.8	3.0-22								

Figure 7. Mapping Table for Site Adaptation Data Element Dictionary and System Database

are used by the NY TRACON software; they are not set or changed by the operational code.

The results of the system and site data parameter analysis was used to determine the parameters to recode in the demonstration system and also to understand how to organize the parameters. By again applying the priorities defined in the statement of work, we were able to determine the parameters that would be recoded. For example, Conflict Alert and MSAW parameters were not recoded, because they related to functions we were not recoding. NY TRACON display unique variables were not recoded because we were using a different situation display. Variables that would map a region of airspace to a display were retained to enable the demonstration system to present tracking data for that airspace on the demonstration system situation display.

A discussion of the organization of the system and site parameters for the demonstration system is discussed in the design section.

3.2 Design

The inputs to the sequential design phase were the outputs of the requirements analysis phase; that is the architecture (concurrent design), the functional requirements document and the data element dictionary.

The sequential software design was completed and baselined in increments. The increments were

- o The Level-1 (or top level) sequential design
- o The Level-2 sequential design.

The software was modeled as functions and/or state machines and refined and constructed algebraically, through the successive replacement of rules and predicates with more concrete and equivalent rules and predicates.

Each package was represented as a state machine prior to recording the design on PDL/Ada. This is illustrated in Figure 8.

The design was recorded as PDL/Ada packages and included the specification of Level-1 packages, their decomposition into Level-2 packages, and the elaboration of the Level-2 packages.

An Ada package comprises a specification, a body and procedures. The specification part defines the behavior -- as a set of operations acting on and encapsulating a set of objects -- of the package at its boundary: users of the package know only that information needed to interface with the package. The body of the package defines the packages internal behavior, and is

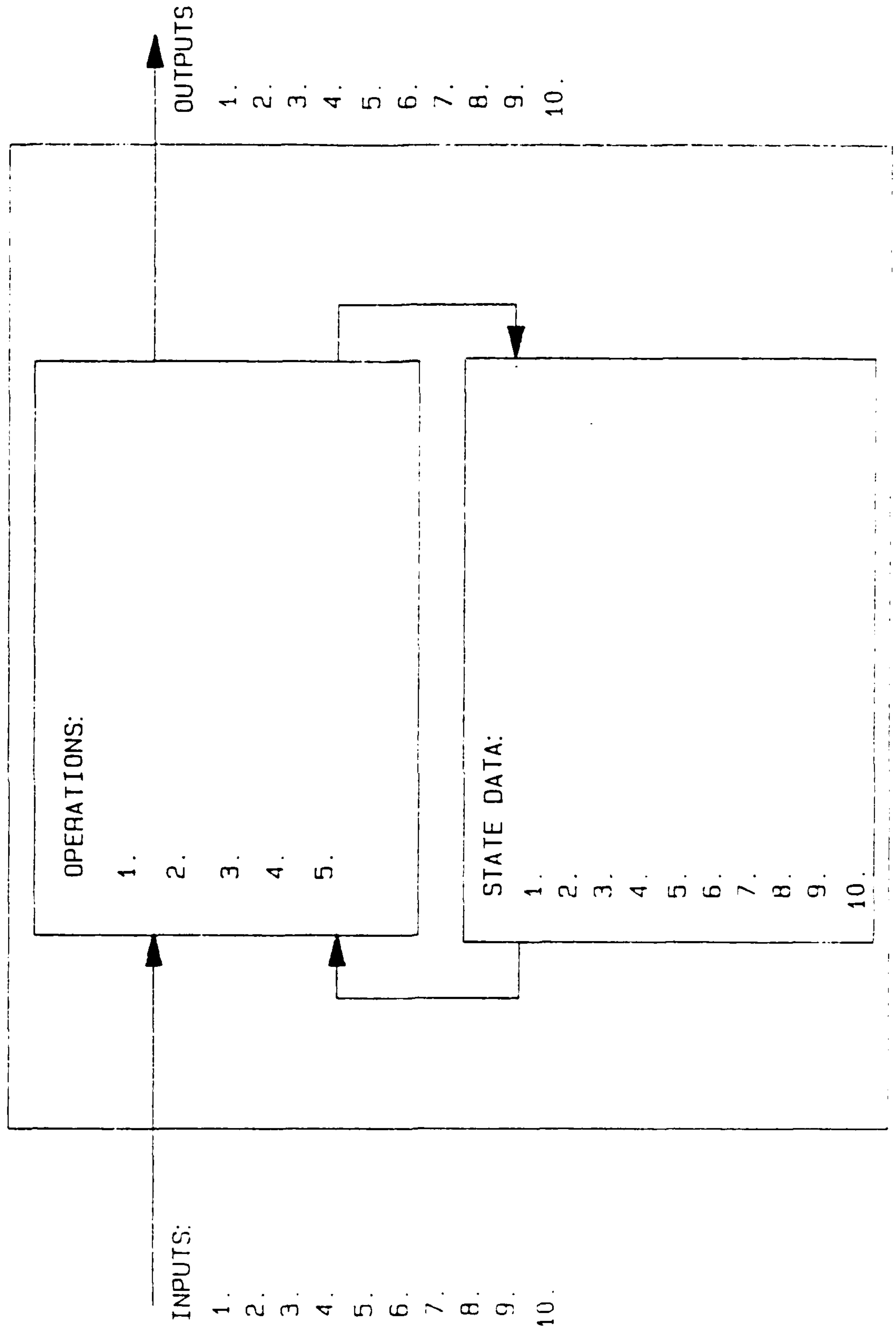


Figure 8. State Machine Diagram

typically a refinement of the specification; the procedures elaborate the operations.

For each online Level-1 Ada package, the template shown in Figure 9, when completed, represents a dispatchable task. To separate the issues of concurrent design from those of sequential design, the online Level-1 packages used a gateway package to record the interface with the applications services. The gateway was specified in a package associated with, but separate from, the Level-1 package for which it is providing the services.

The scope of the Level-1 design was the specification, in Ada, of each abstract data type, at a level that could be verified to be complete and correct by the entire design team. The template used to generate the Ada Level-1 design is shown in Figure 10. The scope of the Level-2 design was the decomposition of the Level-1 Ada packages into Level-2 packages and their elaboration. The template used to generate the Ada Level-2 design is shown in Figure 11.

The paragraphs below define the detailed rules that were followed in using PDL/Ada in each of the two levels of design.

For online packages, the Level-1 sequential design was recorded in an Ada package that was constrained to the Ada specification part only (type definitions, state data and initialization, and operations, called procedures in Ada). Data type packages (see below) were used to define data types. Gateway packages were used to represent the Level-1 package concurrent design, if the Level-1 package was online. For offline packages, such as CDR Editor, there was no gateway package, and the designer had the option of defining data types in a separate package.

3.2.1 Common Data Types

Several PDL/Ada packages were defined to centralize data types:

- TDGLOBAL - containing type definitions that are required by more than two Level-1 packages. An example would be a type that enumerated the names of all the tasks on the system.
- TDCDRMSG - containing type definitions for all messages that are used for CDR format information
- TDSENDS - containing type definitions for records that are passed between the system initialization and termination module and the other operational tasks.
- TCyyxx$$ - (where yy and xx represent the Level-1 package identifiers of the sending and receiving packages) containing the types for messages that are passed between tasks by SEND and are not included in TDSENDS or TDCDRMSG.

```
------------------------------------------------------------------------
-- TMXX$$$$
------------------------------------------------------------------------
-  This member defines the Level-1 sequential design package         --
-- for XXXX.     Associated Level-1 packages are TDXX$$$$            --
-- the Level-1 state data type package, and TGXX$$$$ the             --
-- gateway package at Level-1 design.                                --
------------------------------------------------------------------------
--                        SPECIFICATION                              --
-- TMxx$$$$

--<functional commentary for package.>
package XXXX is
------------------------------------------------------------------------
--Definition Section
------------------------------------------------------------------------
with xxxx_STATE; use xxxx_STATE; --state data types for xxxx
--note: xxxx_STATE is in member TDxx$$$$
------------------------------------------------------------------------
--Intended State Machine Section:
-- State Data:
  DATA1 : state type 1;           --commentary
  .
  DATAN : state type n;           --commentary

-- State Initialization
  DATA1 := value or state;        --commentary
  .
  DATAN := value or state;        --commentary
------------------------------------------------------------------------
-- TRANSITION FUNCTIONS:
------------------------------------------------------------------------
  -- TMxx$$01
  --<one entry for each visible procedure>-----
  --<function of complete procedure.>
  procedure PROC1 (X1 : DATATYPE1);

    -- logic function step
    -- SEND (parm list)     description of send
    -- logic function step
   end PROC1;
------------------------------------------------------------------------

end xxxx;
```

Figure 9. TEMPLATE for Level-1 Sequential Design

```
---------------------------------------------------------------------
-- TDXX$$$$
---------------------------------------------------------------------
--   This member defines the Level-1 state data types for XXXX.     --
--   Associated Level-1 packages are TMXX$$$$ - the Level-1         --
--   sequential design package, and TGXX$$$$ the gateway            --
--   package at Level-1 design.                                     --
---------------------------------------------------------------------
--                 STATE DATA TYPE SPECIFICATION                    --
---------------------------------------------------------------------
--<function commentary.>

package xxxx_STATE is
--Definition Section:
  with yyyy_TO_xxxx;          -- use the interface package defined for
  use  yyyy_TO_xxxx;          -- xxxx and yyyy (TCYYXX$$) if required

  type BUILDTYPE1 is ......;               --commentary
  .
  type BUILDTYPEN is ......;               --commentary

  CONSTANT1 : constant type := value1;     --commentary
  .
  CONSTANTN : constant type := valuen;     --commentary

  type STATE TYPE 1 is  .....;             --commentary
  type STATE TYPE 2 is  .....;             --commentary
  .
  .
  .
  type STATE TYPE N is  .....;             --commentary

end xxxx_STATE;
```

Figure 10. TEMPLATE for Level-1 State Data Package

```
----------------------------------------------------------------------
-- TMXXYY$$
----------------------------------------------------------------------
--                                                                  --
--   This member defines the Level-2 sequential design package      --
--   for XXXX_YYYY. Associated Level-2 package  is  TDXXYY$$.       --
----------------------------------------------------------------------
--                          SPECIFICATION                           --
----------------------------------------------------------------------
-- TMxxyy$$
--<functional commentary for package.>
package XXXX_YYYY is
----------------------------------------------------------------------
--Definition Section
----------------------------------------------------------------------

with xxxx_yyyy_STATE; use xxxx_yyyy_STATE; --state data types
--note: in member TDxxyy$$
----------------------------------------------------------------------
--Intended State Machine Section:

-- State Data:
  DATA1 : state type 1;          --commentary
  .
  DATAN : state type n;          --commentary

-- State Initialization
  DATA1 := value or state;       --commentary
  .
  DATAN := value or state;       --commentary
-----------------------------------------------------------------------
-- TRANSITION FUNCTIONS:
-----------------------------------------------------------------------

  -- The procedures here are the visible procedures only.  The TMxxyyzz
  -- procedures encompass all the procedures, including the hidden ones.

  -- TMxxyy01
  --<function of complete procedure.>
  procedure PROC1 (X1 : DATATYPE1);

    -- function statement that summarizes the processing of
    -- this procedure; e.g., A =: MAX(B,C) describes the
    -- determining of the greater value -- from B and C --
    -- and assigning it A

   end PROC1;
    .
end xxxx_yyyy;
```

Figure 11. TEMPLATE for Level-2 Sequential Design

- TD000000 - a collection point for other data type packages, used as an index, but not referred to by other packages.

3.2.2 Gateway Packages

With the exception of Application Services, each online Level-1 Ada package, in implementation, represents a dispatchable task. The portion of the package that interfaces with applications services to receive work and establish the environment for the application is defined as a gateway.

The gateway was specified in a package associated with, but separate from, the Level-1 sequential design specification for which it provides the services.

The gateway maps the valid commands to the procedures defined in the Level-1 specification. Responses to a conversational send do not appear in the gateway.

3.2.3 Using the PDL/Ada Body to Map Level-1 to Level-2

At the conclusion of the Level-1 design, designers completed the body portion of the Ada package. The body was used to identify the mapping of Level-1 packages to Level-2.

3.2.4 Level-2 Design in PDL/Ada

At least one Level-2 package was defined for each Level-1 package.

The decomposition from Level-1 packages to Level-2 was dictated by the decomposition of the state data space; the decomposition was object- oriented.

If a Level-1 sequential design package decomposed one-to-one to Level-2 (because its state data space was sufficiently small), the designer created a Level-2 sequential design specification, and copied the Level-1 sequential design specification as the foundation for further elaboration. Offline support programs decomposed one-to-one.

When the relationship from Level-1 to Level-2 was one-to-many, the designer created a Level-2 sequential design package for each decomposed object in the Level-1 state data space.

The mapping of Level-1 objects and/or operations to Level-2 packages was recorded in the Level-1 body.

The operations defined in the Level-2 sequential design specification were elaborated in the procedures part of the Ada package.

Each procedure defined, at a minimum, the inputs and outputs, and function rules describing the expected behavior of the procedure. The internal (procedure) variables were defined if used in the function rules; if a function rule refers to identifier "a", "a" was declared. Types required only within a procedure were specified within the procedure.

The body of Level-2 procedure packages, the template which is shown in Figure 12, listed the entire set of procedures including the visible operations and the hidden procedures (housed entirely within the Level-2 package).

3.2.5 Data Base Design

As part of the refinement of the software architecture and the two levels of sequential design, and prior to implementation, the strategy for partitioning and initializing the overall software data base and for building the operational and support subsystems was defined.

The requirements analysis section described the creation of two data element dictionaries, one for operational system data elements and the other for system and site parameter data elements. The analysis also included analysis to determine if the data element was applicable to the demonstration system.

In the design phase, the data base effort determined how to represent the data elements in the demonstration system so that the recoded software could access the data.

For the most part, the operational system data elements were recoded to reside in the same structure as in the NY TRACON system. These data elements were in the Central Track Store (CTS), Target Report Store (TRS), Radar Only Target (ROT) Table and the Beacon Only Target (BOT) Table.

The system and site parameters presented a challenge in definition, organization, and setting. The system and site DED referred to each parameter by the name given it in the CPFS, which is the name given to each individual data item. If a data item had 150 values, it was defined using 150 unique names. We needed to develop a scheme for defining the data items without a proliferation of names that were unique to NY TRACON. In addition, these data items were assembled into data elements in the operational parameter DED using macros in specially coded data base members. We needed to develop a more generic way of defining the data for the operational system. Finally, some data items were assembled without consideration for related types of items. If there were a number of items, say 10, that described keyboard information, there might be 10 arrays of data, with each element of the array describing the value for one keyboard. We

```
--------------------------------------------------------------------------
--                              PROCEDURE
--------------------------------------------------------------------------

    -- TMxxyyzz

    procedure PROCEDURE_LONG_NAME
              (PARAMETER_1                : in    PARM_1_TYPE;
               PARAMETER_2                : in    PARM_2_TYPE;
               PARAMETER_3                : out   PARM_3_TYPE);
    --
    -- State the intended function
    -- used in the procedure statement
    -- of the specification (e.g., A := MAX(B,C))
    --
    -- Used by: PROCEDURE_LONG_NAME                          -- TMxxyyzz
    --          PROCEDURE_LONG_NAME                          -- TMxxyyzz
    --
    -- Uses   : PROCEDURE_LONG_NAME                          -- TMxxyyzz
    --          PROCEDURE_LONG_NAME                          -- TMxxyyzz
    is
                                  -- type local data
       LOCAL_1_TYPE = INTEGER;
       LOCAL_2_TYPE = BOOLEAN;
       LOCAL_3_TYPE = INTEGER;

                                  -- declare local data
       LOCAL_DATA_1 : LOCAL_1_TYPE;
       LOCAL_DATA_2 : LOCAL_2_TYPE;

    begin
                                  -- intended function
       if
         A = B
       then
         C := D;
       else
         C := E;
       end_if;
                                  -- intended function
       while
         A  = C
       loop
         A := A + 1;
         B := B - 1;
       end_loop;

    end PROCEDURE_LONG_NAME;
```

Figure 12. TEMPLATE for Level-2 Procedures

needed to group data that described different aspects of one object together, for ease of reference.

To address the type items described above, we implemented the following:

- o All items that described an entity, say a keyboard or display, were all grouped together in one entry of an array.

- o A name was given to each item that it could be referenced by the operational code. The name uniqueness from the current NY TRACON system was removed.

- o As each set of data items was defined, it was initialized by defining the constants in the data item. The data was preset using Pascal/VS structured constants.

Top level software design was completed prior to the start of detailed low level design. To ensure that the nine-month schedule was met, the low level design period overlapped the development of build 1 modules, although all low level designs were completed prior to the start of build 1 software integration and testing. All PDL modules were inspected and documented at each level.

This design approach, including the use of PDL/Ada, encouraged more organized structure and greater modularity, aided identification of dependencies, state and local data, and streamlined access to tasks and procedures. It also facilitates software development in any HOL, and on any contemporary operating system and computer.

3.3 Code and Development Test

The New York TRACON demonstration system was implemented by converting the Level-2 PDL/Ada to Pascal/VS.

The implementation life cycle activities and products were

a. Source code generation

b. Generating unit/string test plans

c. Inspecting the source code and test plans and reporting the results

d. Conducting unit tests

e. Conducting string tests

f. Creating builds

g. Conducting build tests from a structural point of view (in preparation for formal software integration and testing)

h. Controlling the configuration of the software at each node

i. Fixing errors and retesting the software at each node.

The New York TRACON demonstration software was built incrementally. The sequential design unfolds from Level-1 packages, through Level-2 packages, and procedures. The implementation starts with units (procedures may comprise one or more units), and proceeds through strings and builds. There were multiple builds and each build was tested while the successive build was implemented.

The following subsections describe the procedures for defining, documenting, packaging, inspecting and testing software units, and configuring them (and testing the configurations), until a completed software build resulted.

SOURCE CODE GENERATION

The composition of Level-2 packages (into a single Level-1 package) represent a Pascal/VS load module; and the unit of execution in Pascal/ VS is a load module with a single entry and exit.

Figure 13A and 13B illustrates the process of converting a PDL/Ada design unit to a Pascal/VS procedure.

The New York TRACON software was coded in Pascal/VS and, where necessary, in S/370 assembly language.

The official reference guide for Pascal/VS is the "Program Offering, Pascal/VS Language Reference Manual, Program Number 5796-PNQ."

A single member, named for each Level-1 package (TMxx$$$$), in the appropriate build 1 development string library, e.g., TR0501.DDISPLAY. PASCAL, contain a list of the compilation (and assembly) units that compose that Level-1 package.

Each compilation unit was identified by a descriptive name (e.g., ITERM); the descriptive name was identical to a PDL name, if there is a PDL counterpart. If a compilation unit can be traced to an ULTRA name (representing a program or data that performs the same function), the ULTRA name was used.

The name of a compilation unit, a Pascal/VS main program, a Pascal/VS segment, or an Assembly Language source module, is identical to the name of the library member in which it resides.

PDL/Ada	Pascal/VS
`` Package TMxx$$$$ is --definitions with TDGLOBAL; with TDSEND; with TDxx$$$$; with TM$STATE; --TM$STATE contains the --STATE_DATA package --STATE type STATE_DATA is record field1 : atype; field2 : btype; . . end record ; var TASK_STATE : STATE_DATA ; INPUT : COMMUNICATION_PACK; ```	``` program TMxx$$$$; %include TDGLOBAL; %include TDSEND; %include TDxx$$$$; (* TDxx$$$$ is a member in either TR0501.Dyyyyyy.MACLIB or TR0501.DCOMMON.MACLIB and contains the constant and type declarations *) type STATE_DATA = record field1 : atype; field2 : btype; . . end; (* record STATE_DATA *) STATE_PTR = -> STATE_DATA; var TASK_STATE : STATE_PTR; SIZE : INTEGER ; INPUT : -> COMMUNICATION_PACK; %include TMSTATE ; %include TM01$$$$;{SEND/RECEIVE} %include TMyy$$$$; %include TMzz$$$$; {TMyy$$$$ and TMzz$$$$ are member in either TR0501.Dyyyyyy.MACLIB or TR0501.DCOMMON.MACLIB and contain external procedure declarations} ```

Figure 13A Converting PDL/Ada to Pascal/VS

<table>
<tr><th>PDL/Ada</th><th>Pascal/VS</th></tr>
<tr><td><pre>
procedure Main ;

  RECEIVE ;
  case INPUT.CMD is
    when X => call A;
    when Y => call B;
        .
        .
  end case;
end TMxx$$$$;
</pre></td><td><pre>
begin   { Main Program   }
  TASK_STATE := GetState ;
  if TASK_STATE = nil
  then
    begin
    SIZE := SIZEOF(TASK_STATE)
    GetNewState(TASK_STATE,SIZ
    (* initialize TASK_STATE *
    end ;
  RECEIVE(INPUT) ;
  case INPUT->.CMD of
    X : A;
    Y : B;
      .
      .
  end; { case }
end; {Main program TMxx$$$$}
</pre></td></tr>
</table>

Figure 13B Converting PDL/Ada to Pascal/VS

Under each compilation unit name are listed the visible procedures and functions; the prologue of each compilation unit contains a list of all units composing the compilation unit.

Program preambles define the functional, structural and performance attributes and summarize the internal behavior of each unit, such that future maintainers of the source have no difficulty understanding its contents, nor its relationship to the other system software.

Developmental history and problem resolution information is not included with the source code; it is produced automatically by the automated program trouble report system and the accounting system.

The rules for packaging of Pascal/VS units and Assembly units are slightly different and will be discussed separately. The following definitions are common to both.

- Entry point - a location in a module to which control can be passed from another module or from the control program.

- Execution unit - object code that can be executed on a computer. (A load module.)

- Link edit - process of combining separately compiled object modules into an executable load module. The output of a link edit is a load module; symbolic cross references among object modules are resolved during link edit.

- Load Module - object code in a format suitable for execution; the output of a link edit.

- Loader - combines the basic functions of a linkage editor with the execution of a program. Used during testing of a load module.
- Object module - the output of a compiler or an assembler. (In our application, the Pascal/VS compiler and Assembler H.) Object modules are input to the linkage editor or loader.

- Program entry point - address in the load module to be given control by the control program whenever the load module is executed.

The templates were used for defining PASCAL/VS and assembly language units. Figures 14A and 14B contains the Pascal procedure or function template used in code development.

```
{   ------------------------------------------------------------------ }
{                   PROCEDURE or FUNCTION                              }
{   ------------------------------------------------------------------ }
{   Procedure or Function Name  : aaaaaa                               }
{   ------------------------------------------------------------------ }
{   Descriptive Name:    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa             }
{   ------------------------------------------------------------------ }
{   Function:                                                          }
{                                                                      }
{    (Short summary of function)                                       }
{                                                                      }
{   ------------------------------------------------------------------ }
{   Date Of Implementation:    mm/dd/yy                                }
{   ------------------------------------------------------------------ }
{   Input:                                                             }
{   Calling Sequence Input:                                            }
{                                                                      }
{        Input                                                         }
{      Parameter           Desc. Of Input Parameter and Variable Name  }
{      ---------           ------------------------------------------  }
{        IPARM1            (DESCRIPTION OF INPUT PARAMETER 1)          }
{        IPARM2            (DESCRIPTION OF INPUT PARAMETER 2)          }
{          .                          .                                }
{          .                          .                                }
{        IPARMN            (DESCRIPTION OF INPUT PARAMETER N)          }
{                                                                      }
{      Note: If a parameter is both input and output, describe in both }
{  ------------------------------------------------------------------- }
{   Output:                                                            }
{   Calling Sequence Output:                                           }
{                                                                      }
{       Output                                                         }
{      Parameter           Desc. Of Input Parameter and Variable Name  }
{      ---------           ------------------------------------------  }
{        oparm1            (Description of output parameter 1)         }
{        oparm2            (Description of output parameter 2)         }
{          .                          .                                }
{          .                          .                                }
{        oparmn            (Description of output parameter n)         }
{   ------------------------------------------------------------------ }
{   Output Messages (SENDs):    (If none, indicate "none")             }
{   ------------------------------------------------------------------ }
{   Assumptions/Unresolved issues:   (If none, indicate "none")        }
{   ------------------------------------------------------------------ }
{   Calling Modules : (Include a list of procedures that call )        }
{   ------------------------------------------------------------------ }
{   Called Modules (Internal):  (Include a list of procedures that     }
{                                this program calls that are internal  }
{                                to this procedure)                    }
{   ------------------------------------------------------------------ }
```

Figure 14A Procedure or Function Template used in Code Development

```
{   Called Modules (External):  (Include a list of subprocedures that }
{                                this procedure calls that are          }
{                                external to this procedure)            }
{   --------------------------------------------------------------------}
{   Date Of Last Modification:  mm/dd/yy                                }
{   --------------------------------------------------------------------}
{   ******************************************************************* }
     %page

    procedure PROCEDURE_LONG_NAME
              (PARAMETER_1                    :   PARM_1_TYPE;
               PARAMETER_2                    :   PARM_2_TYPE;
               PARAMETER_3                    :   PARM_3_TYPE);

         OR - OR - OR - OR - OR - OR - OR

    function  FUNCTION_LONG_NAME
              (PARAMETER_1                    :   PARM_1_TYPE;
               PARAMETER_2                    :   PARM_2_TYPE;
               PARAMETER_3                    :   PARM_3_TYPE) :
               return_type ;
 {                                                            }
 {      Used by: PROCEDURE_LONG_NAME                          }
 {               PROCEDURE_LONG_NAME                          }
 {                                                            }
 {      Uses   : PROCEDURE_LONG_NAME                          }
 {               PROCEDURE_LONG_NAME                          }

    type
                                    -- type local data
       LOCAL_1_TYPE = INTEGER;
       LOCAL_2_TYPE = BOOLEAN;

    var                             -- declare local data
       LOCAL_DATA_1 : LOCAL_1_TYPE;
       LOCAL_DATA_2 : LOCAL_2_TYPE;

    begin                             { begin procedure aaaaaaa  }
       .
       .
    end ;                             { end procedure aaaaaaaaa  }
```

Figure 14B Procedure or Function Template used in Code Development

GENERATING UNIT/STRING TEST PLANS

Unit and string test plans were written and inspected with the source code. A test plan template was used.

INSPECTING SOURCE CODE and TEST PLANS and REPORTING RESULTS

All software units were formally inspected. Figure 15A and 15B represents the template for the FAA TRACON Quality Analysis Report.

a. The entry criteria to each inspection are

- Completed design, code (compiled) or test cases without syntax errors;

- Make available at the inspection the preceeding level of specification: the specification tree is

Development	Testing
- Requirements analysis	
- Level-1 Ada packages	- String tests
- Level-2 Ada packages	- Unit test planning
- Source code	- Unit test cases

- The timing for inspecting test plans and test cases may vary, but the test exhibits must be inspected prior to the running of the tests.

- The size of the material must be small enough to complete the inspection in 3 hours.

- The moderator, author, presenter and inspectors must have inspected all material prior to the meeting and should report as many errors as possible to the author prior to the meeting.

- If any of the above criteria are breached, the moderator may postpone the meeting.

b. To exit an inspection the inspected material must have satisfied the inspectors that the design, code or test performs the function specified by the previous level specification, with a minimum of rework. The moderator is responsible for determining whether the material must be re-inspected.

```
------------------------------------------------------------------------
| A.  TYPE OF REPORT                                                    |
------------------------------------------------------------------------
|  Type (TLD,LLD,CD,TC)        ==> CD
|  -----------------------------------
|  | TLD: top-level design inspection |
|  | LLD: Low-level design inspection |
|  | CD:  Code inspection             |
|  | TC:  Test case inspection        |
|  -----------------------------------
------------------------------------------------------------------------

------------------------------------------------------------------------
| B.  ADMINISTRATIVE DATA                                               |
------------------------------------------------------------------------
|  Date of Inspection          ==> October 31, 1986
|  Time                        ==> 9 AM
|  Location                    ==> 865 2D-26
|  CPCI/CPC                    ==> 03
|  Build Number                ==> 01
|  Modules or Test Case No.s   ==> TM01$$$$, TM02$$$$, TM03$$$$
|                              ==> ITERM, SEND, RECEIVE modules
------------------------------------------------------------------------

------------------------------------------------------------------------
| C.  INSPECTION INVITATION AND PREPARATION                             |
------------------------------------------------------------------------
|  Inspection Attendees        ==> Moderator:
|                              ==> Author   :
|                              ==> Presenter:
|                              ==> Inspector:
|                              ==> Other    :
|                              ==> Other    :
|                              ==> Other    :
|  Re-inspection? (Yes/No)     ==> No
|  Labor-hours to prepare      ==> 14
|  Labor-hours to inspect      ==> 12
|  Labor-hours to rework       ==>  0
|  Clock-hours of  meeting     ==>  3
------------------------------------------------------------------------
```

Figure 15A FAA TRACON SOFTWARE QUALITY ANALYSIS REPORT

D. METRICS (NOT USED FOR TEST CASE INSPECTIONS)

Design:	lines of PDL	==> 100	Estimated SLOCs ==> 500
	lines of comment	==> 20	
	total lines	==> 120	
Code:	lines of prologue	==>	
Only	lines of instruc.	==>	
	lines of data	==>	
	lines of comments	==>	
	total lines	==>	

E. ERROR REPORT

Categories of Errors		Major errors	Minor errors
Standards violations	==>		
Inadequate requirements	==>		
Interface error	==>		
Type specification error	==>		
Incorrect logic	==>		
Incorrect data declaration	==>		
(Test cases only)			
Inadequate test plan	==>		
Inadequate test coverage	==>		

Re-inspection required ==> No

Moderator Date

Figure 15B FAA TRACON Quality Analysis Report

c. Major errors are errors in:

- standardized names and identifiers
- intended function statements
- type specifications (arguments, data or functions)
- control structures
- data declarations
- test case procedures and test data

d. Minor errors are errors in

- form (alignment, readability, etc.)
- commentary

CONDUCTING UNIT TEST

The entry and exit criteria for this node (Unit exit -- String entry) are:

- o No compilation errors
- o Inspections complete
- o Unit testing complete

Test drivers were used that incorporated certain debugging routines, such as dump a CTS track file, print a boolean value.

CONDUCTING STRING TESTS

The entry and exit criteria for this node (String exit -- Build entry) are:

- o No compilation errors
- o Inspections complete
- o String testing complete

The string test drivers used were similar to the unit test drivers. String testing executed a number of related functions. Another type of string testing involved using RETRACK under VM to provide the test inputs for TRACK string testing.

CREATING BUILDS

To create a build all development work was stopped and prepared for promotion. These procedures were followed.

If some members have been promoted, run a SUPERCOMPARE of the Dlibs to the Rlibs. If the only changes are debugging OLPRTs, then no repromotion is required. If source changes were found, PTR numbers were assigned following the PTR procedures.

CONDUCT BUILD TESTS

The build was link-edited then tested using a GFE tape. The build test was successful if the entire tape was processed and the results were expected.

CDR Editor was performed on the recorded data to verify the results of the test.

4.0 Verification Methodology

The process for verification of the demonstration system's performance, standards, integrity, software, output reports (specifically CDR Editor output report), and display outputs was a multi-tiered methodology. This methodology was defined, instituted, and enforced by developers, build committees, string leaders, software engineers, the independent Software Integration and Testing (SWIT) team, and the FAA and its representatives (SEI).

The primary objectives of this verification process was not only to attain the same performance from the reengineered demonstration system functions, as the respective N.Y. TRACON A5.04 functions, but also to maintain functional equivalency, wherever applicable. In addition, the demonstration system was tasked to maintain the tried and tested Tracking algorithms, to generate a CDR Editor output report identical in content to the N.Y. TRACON A5.04 CDR Editor output report (using the input file generated by the ARTS system), and to demonstrate this equivalency visually through the situation display developed for this project.

This section contains the chronological sequence of activities that comprise the processes and methodology inherent in the verification of the demonstration system.

Step #1A - Initial Verification Tools and Standards
(Section 4.1)

Software verification is an ongoing process which commences at proposal generation, with the definition of processing methods, plans, and approaches to achieve the contractual objectives, and continuing through to the design, development, and implementation phases. The demonstration system's team, in its approach to software verification, incorporated several aspects of this verification approach prior to software development or formal systems design.

Step #1B - Mapping of the MPE services to MVS/RTX
(Section 4.1.1)

This initial effort, completed prior to contract start-up, and exhibited within the N.Y. TRACON Operational Demonstration of Program Recoding Technical Proposal resulted in the mapping of the MPE services to MVS/RTX.

Step #1C - Charting the Expected Behavior of the Software (Section 4.1.2)

During the requirements analysis phase the demonstration system's team reviewed and analyzed the current ARTS IIIA system and all available ARTS IIIA document (GFE) resulting in the definition and documentation of the expected behavior of the demonstration system's software.

Step #1D - Formulation of the Software Measurement and Performance Standards (Section 4.1.3)

These guidelines and measurement criterion were developed to standardize all software, deliverables, internal products, and program status documents, and served as a benchmark from which all work products were measured.

Step #1E - Defining the Development Processes and Procedures (Section 4.1.4)

This effort resulted in the definition of a single set of development processes and procedures for all program products, and it was instituted through the use of templates, outlines, and summaries.

Step #2 - Software Build and Integration Testing (Section 4.2)

Multiple layers of testing, including Unit, String, Build, Benchmark, and SWIT testing were instituted to verify the integrity of the software developed, redesigned, and/or reengineered.

Step #3 - Build Testing (Section 4.2.1)

The software development was organized into three builds. Each build included four levels to which the testing process was directed to verifying. The four levels of testing were unit, string, benchmark, and SWIT.

Step #4 - Quality Assurance (Section 4.2.2)

Quality assurance procedures were instituted to enforce compliance to baselines and standards, and to monitor functional equivalency.

Step #5 - Reviews (Section 4.2.3)

Inspections, re-inspections, program and technical reviews, and demonstrations were also conducted to monitor adherence to project standards, and to verify the integrity of the software.

Step #6 - Traceability (Section 4.3)

A major contributant to the verification process was the development of traceability matrices that depicted the mapping of the ULTRA source code to the PDL/Ada software design, and from PDL/Ada to the Pascal/VS HOL.

Step #7 - Levels of Traceability (Section 4.3.1)

Three levels of traceability were developed for the representation of the demonstration system's software. The categories of traceability were, directly traceable, redesigned category, and the category that were implemented differently (due to differences in hardware configuration and/or operating system).

Step #8 - Maintaining Traceability (Section 4.3.2)

Based on contractual requirements, operating system differences, and architectural differences, some functions were required to be directly traceable and functionally equivalent, while others maintained only assemblance of traceability. This sub-section elaborates on these differences.

Step #9 - Traceability Matrix (Section 4.3.3)

This sub-section describes the components of the traceability matrix, how it was developed, and the information sources used to generate it.

Step #10 - CDR Editor Listing Output Validation (Section 4.4)

The main component of the verification process was the comparison of the demonstration systems CDR Editor output report to the ARTS IIIA CDR Editor output report. This verification was completed first by the demonstration system's team and then by the FAA.

4.1 Initial Verification Tools and Standards

Prior to any software development, some of the activities, reviews, and analysis resulted in the development of viable tools, standards, and guidelines that also contributed to software verification. They included :

- Mapping of MPE services to MVS/RTX
- Charting the expected behavior of the software
- Formulation of the software measurement and performance standards
- Definition of the development processes and procedures

4.1.1 Mapping of the MPE Services to MVS/RTX

This effort, completed prior to contract start-up, and presented in the N.Y. TRACON Operational Demonstration of Program Recoding Technical Proposal, mapped the critical components and services provided by the ARTS IIIA system (See Figure 3 Mapping of the MPE Services to MVS/RTX). The mapped services and components were developed and defined by the architectural committee as the minimal requirements to be satisfied by the demonstration system.

This approach ensured that the applicable subsets of the reengineered system was functionally equivalent to its respective subset in the current ARTS IIIA operational system. It also made the application work much simpler by helping to identify interrelationships and interfaces, and this relieved the developers from real-time and MVS/RTX considerations, and allowed them to concentrate on their specific application areas. This approach also provided a benchmark from which the demonstration system's components and services were measured during the development, design, and implementation phases.

4.1.2 Charting the Expected Behavior of the Software

To analyze and verify the integrity, accuracy, completeness, and consistency of the software, it was necessary to define and document the expected behavior of the software (See Appendix A System Parts and Their Work). The expected behavior of the operational software was determined by an analysis of system work, system functions and their relationship to the work, the decomposition of parts, tasks and concurrency, communications between tasks, data bases, and data coherency.

This effort was conducted during the requirements analysis phase of the contract by the architectural committee. The current ARTS IIIA system, all available ARTS IIIA documents (GFE), and the proposed demonstration system requirements and objectives were reviewed and analyzed in order to chart the expected behavior of the demonstration system software.

The benefits derived from this effort contributed to the definition of the application packages, characterization of the flow of work through the system, well defined task interfaces, and the maintenance of functional equivalency (where applicable).

4.1.3 Formulation of the Software Measurement and Performance Standards

The major guideline for all work products developed for the demonstration system was that all work products, including the software, deliverables, program status documents, and internal work products must be measurable. To achieve this objective, the demonstration system's team developed a single set of program controls based on government standards, to monitor and control the project. These controls were tailored specifically to a project of short duration. The controls and measurement criterion were :

- o The correctness, at each level of abstraction, including the architecture and top level design, detailed design, and source code.

- o The traceability (and correctness of the trace)

  1) to the requirements;
  2) between levels of development (design--source code, etc.);
  3) between the converted software and the GFE ULTRA software.

- o The development and informal verification of the architecture and the top level design.

- o The development and correct execution of software modules.

- o The development and running of unit and string test cases.

- o The completion and correct execution of software build.

- o The development and running of software integration test cases.

- o The completion of the software integration and testing.

- o The number and severity of design issues.

- o The number and severity of software errors.

- o The successful controlling of the software configuration.

- o The measurement of all deliverables, internal products, and program status documents, in terms of their completeness, coherency, clarity, correctness, precision, and conciseness, as agreed to by the contractors' internal quality review and/or by the FAA.

Detailed development milestones were also developed and monitored for adherence to overall project schedule. A project work breakdown structure (WBS), based on the software system life cycle activities was also developed, and it enabled management to track cost/schedule performance for the program, and to identify and/or forecast possible problem areas, bottlenecks, and dependencies. Initial source lines of code (SLOC) estimates, by computer program configuration item (CPCI), were presented in the technical proposal. They provided management with a cross-reference for each program product developed or reengineered. Configuration control was maintained by the establishment of two separate levels of product control : a baseline level and a development level.

4.1.4 Defining the Development Processes and Procedures

The demonstration system team selected a representative configuration committee who was primarily responsible for the definition of the development approach and standards. The committee formulated a single set of development standards for each project development phase, which were transmitted to the developers via electronic public libraries. They included templates, outlines, and summaries that aided design, coding, and testing (See Figure 9 Template for Level-1 Sequential Design).

The committee, headed by a system engineer, verified that the standards were met. Deviations were permitted only after a thorough review was completed by the committee and also the project's upper management.

The development approach chosen, was the incremental build approach, or building on tested and approved modules. This ensured that the system was built in small manageable units. This approach also facilitated the software integration and testing (SWIT) activities, which were performed concurrent with module development and unit testing. It allowed SWIT and the developers to test each build while the successive build was being implemented.

4.2 Software Build and Integration Testing

Several layers of testing were implemented to verify the integrity, accuracy, guidelines, standards, and functional equivalency of the recoded, developed, and/or the redesigned demonstration system software. They included :

- Build Testing
- Quality Assurance Testing
- Design and Code Reviews

4.2.1 Build Testing

The software development was organized into three builds. Each subsequent build was directly additive to the previous. Each build was required to use as input the operational continuous data recording input file and provide evidence of successful operation. To accomplish this, the first build established the system architecture, primary services to the application programs, the system driver (RETRACK), the concurrent portion of each application package (GATEWAY), and the tape conversion program. The build performed system and task initialization, went into steady state processing, and successfully terminated each task. During steady state processing, CDR input was read by RETRACK and sent to the appropriate task gateway, using application services.

The second build added the CDR editor and extraction functions, established the link to the display (displaying the map and range rings), established the data base and common routines, PSRAP, and the syntax verification and parsing portion of the keyboard and interfacility functions. Build 2 was validated by analyzing the CDR extraction produced by the run. The extraction output contained sector marks, keyboard, and interfacility messages.

The third build added the tracking processing and display functions, and processing for keyboard and interfacility messages. Full display, CDR extractor output, application online printouts, and RTX log messages were used for problem analysis and test verification.

The testing process was directed towards verification of each build, and included four levels :

- o Unit
- o String
- o Benchmark
- o SWIT

Unit testing was performed primarily by the developers of the packages (tokens,modules). The packages were first compiled and tested under the VM/CMS operating system as independent entities, to eliminate syntactical errors. After a successful compilation was achieved, the package was then recompiled while incorporating gateway, application, and dependent packages (using Pascal/VS %Include), which aided in eliminating most logical errors.

A successful compilation of the incorporated packages was the final phase of a unit test.

String testing, the testing of a combination of logical or interrelated units, was performed by appointed string leaders, in corporation with developers of the interrelated packages. Each string leader was responsible for generating a driver, where necessary. The generation of data drivers, used to exercise the strings, was also the responsibility of string leaders.

All string tests were completed under the VM/CMS operating system.

Benchmark testing, a viable entity that works optimally with an operating system that produces valid data, was also conducted on the demonstration system's software. Benchmark testing was performed to measure the performance of the demonstration system's software against the performance of the respective ARTS IIIA software. This method of testing the system's software used valid and tested strings which were always initiated through the Retrack function. It was designed to test valid data through all components of the strings.

Each level of test used appropriate tools and followed procedures that had been reviewed. Each developer submitted a unit test plan for inspection with the low level design inspection material. Each string leader submitted a string test plan and results to the build coordinator.

The purpose of the unit test was to verify the logical correctness of each module. As modules were verified, they were grouped into larger units for testing. This testing was performed under VM using drivers written to support each separate module. As required, developers developed stubs for applications that they interfaced with, and system services that they required. Data from the operational CDR tapes were analyzed and coded into the drivers to maximize use of real data.

Upon completion of unit tests, string tests were conducted. A string test consisted of several units compiled and tested together. The string tests verified the interface between all modules in a package. String tests were run in an incremental fashion, combining modules until all modules within a package were successfully tested. String tests were run interactively under VM using drivers and special routines that simulated the MVS/RTX services. As with the unit tests, maximum use was made of operational data to assure thorough and accurate testing. String leaders were appointed to oversee successful string test completion.

Before passing a build to SWIT the entire build was tested at the development level under the control of a build leader. The purposes of the build tests were to verify the correctness of the interfaces between the various strings and to ensure that the architecture and use of the system services was still correct. Upon completion of build test, the software was promoted from the development to the release level.

When the build reached the release level, overall system testing was performed by SWIT. SWIT provided detailed guidelines for test execution and clearly documented any discrepancies from the expected results. If a discrepancy occurred, a Program Trouble

Report (PTR) was issued and sent to developers. Responsibility to explain and resolve the PTR was assigned to the developer.

Final system testing was conducted by SWIT. SWIT developed the test plan, build plans, test schedules, and identified all necessary equipment and personnel required for integration and test.

The demonstration system was configured into two distinct components. The first component was developed to run online in real-time, using the MVS/RTX operating system. The second component, containing the CDR Editor and Conversion program, ran offline using the VM/CMS operating system.

During system initiation of the online demonstration subsystem, the Send/Receive utility was directed to transmit initialization messages to all software module message gateways. The Send/Receive module handled all data distribution functions which provided the capabilities for each software module to transmit data to other modules. The message gateway was the central point of entrance and exit for each software module. At each gateway, the data message codes were examined, and the messages were passed on to the respective subroutine for processing.

Data enters the online demonstration system when the Retrack module reads it from the input CDR message file. The application package modules directs the send/receive modules to distribute these messages to 3 software gateways: Keyboard, PSRAP, and Interfacility. The Keyboard, PSRAP, and Interfacility modules process data received from the Retrack module and sends the data to other modules within the system.

Data exits the online demonstration system when the CDR Extraction module extracts CDR data messages from the PSRAP, Tracking, Keyboard, and Interfacility (input and output processing) modules at specified time intervals. This data is retained as output in a CDR Extraction file.

The messages extracted contained a mix of CDR messages, which were identical to the original CDR messages injected into the system by the Retrack module. The Retrack module also provided the capability to fabricate CDR messages. These fabricated messages were, flight data entry messages that were necessary to initiate the track file for targets already within the tracking area at start time. In the ARTS IIIA system, tracks are established when a target initially enters a TRACON area, but in the demonstration system, at system startup the data needed to initiate some tracks were not included on the GFE CDR tapes. The types of CDR messages expected within the CDR Extraction data set included: Sector Time, Target Report, Keyboard, Interfacility, and Radar Only Target Report messages. The messages extracted were identical to the original CDR input data. The fabricated flight data messages were verified by comparing them to the CDR input Tracking messages.

Since the system jobs were run native on the IBM 3083 computer, the Job Control Language (JCL) was submitted using panel driven interfaces. The results of the job run was then routed to the users terminal. After the system run, the CDR Extractor data set was transferred from the IBM 3083 computer and MVS operating system to the IBM 3081 computer under the VM/CMS operating system to facilitate the execution of the CDR Editor and the generation of its hardcopy report. This process was instituted by copying the data set to tape and then uploading it to the IBM 3081 computer system.

The CDR Editor, read the CDR extraction file and generated a CDR messages report that summarized the CDR messages found. It also has the ability to create 4 types of message reports, Interfacility, Keyboard, Auto Function, and Target Report. The data must consist of an Initialization message, which must be the first message, and it must end with a Termination message.

To create an Interfacility or Auto Function message report, the CDR Editor used as input the CDR Extractor data file. All the other messages were then filtered out. To generate a Keyboard or Tracking message report, the CDR Editor also used as input the CDR Extractor data file in the same manner as when creating an Auto Function Report. The Sector Time messages were used to create sensor summary reports for the Keyboard and Tracking reports. The sensor summary reports consisted of a one line tally of all targets detected after a complete sensor sweep of 360 degrees. For the 4 active radar sensors which revolve at differing rates, the target detection count was reported for each separately upon completion of their respective revolution.

4.2.2 Quality Assurance

Cross-checking procedures were followed when implementing the reengineered TRACON software. The developers were responsible for performing unit testing of their respective modules. Cross-checking was instituted to enforce compliance to baselines, standards, and guidelines, and to ensure that functional equivalency was maintained, wherever applicable. The cross-checking was accomplished through design and code reviews performed by build committees. Additional cross-checking was completed across functional lines (e.g.: Tracking function), and instituted by developers of dependent routines, and developers of segments or routines of the respective function.

When a module was ready for promotion to the release level, the developer added an accounting card. The accounting card reflected the build number of the module. The build coordinator checked the accounting card, and if satisfied, directed the developer to complete a promotion form which was approved and signed off by management and the build coordinator. The promotion form was submitted to SWIT, who had the responsibility for promoting the module to the release level. During SWIT testing, if problems were encountered with the modules a PTR was issued. The PTR documented required changes or improvements. If changes were required, management assigned the problem analysis and resolution to a developer. When the developer completed the necessary improvements, a second accounting card, a change update card, was issued. The change update card contained the date and the PTR number, which was issued in sequential order. If the build coordinator approved, the module was again released to SWIT. As each build was completed, all release level members were copied to the next build. This build number was reflected in the accounting card numbers. In this way, a history of each module was maintained, and the module's history could be traced through the accounting file numbers.

4.2.3 Reviews

Design and code inspections were conducted by technical peer groups for each work product developed for the demonstration system. These groups were comprised of, a moderator, an inspector, a presenter, the developer(s), developers of dependent routines, and other relevant personnel. The inspections were formal in nature, and they were standardized through the development of templates specifically designed for the said purpose (See Figure 15A and 15B FAA TRACON Quality Analysis Report).

The inspection team recorded results, examined the code or design, test cases, and qualified the errors inherent in the package being inspected. Re-inspections were conducted by the same inspection team if severe errors were identified.

These inspections were conducted to ensure functional equivalency and accuracy, to enforce project standards, and to verify the technical correctness of the demonstration system software, top and low level designs, and test cases.

Program and technical reviews, and demonstrations were also conducted with the FAA. They were held at FAA headquarters, and at IBM-FSD's Rockville facility on dates mutually agreed upon by the contractors, and the FAA. The topics of the reviews included:

- The status of all software and deliverable work products.
- The cost expended to date.
- The status of the work with respect to the overall schedule and the forecast for completing the work.

- The status of work with respect to the detailed schedules, as requested by the FAA.
- The status of all action items.

Demonstrations, ending with a final demonstration on May 29, 1987, were also conducted. This final demonstration used an FAA provided input file, containing the recorded output of a current FAA N.Y. TRACON ARTS IIIA run. The output from the demonstration run was compared to the N.Y. TRACON ARTS IIIA system generated output, and no unanticipated discrepancies were found. The output was verified by the contractors and the FAA.

4.3 Traceability

One of the major constraints on the development of the demonstration system, was the maintenance, development, and documentation of traceability. For the demonstration system, traceability was maintained between the subset of the ARTS IIIA system ULTRA source code reengineered, the two levels of software design (PDL/Ada), and the recoded, newly developed software, and/or redesigned software in Pascal/VS HOL.

4.3.1 Levels of Traceability

In support of the traceability *requirement*, the demonstration system team developed three categories of traceability, for the reengineered TRACON software. The first category, was that group which was directly traceable. An example of this category was the Tracking modules. The second category, was that group whose basic function was equivalent, but still had to redesigned, either to fit into the system architecture or to accommodate a subset of the complete function being translated. The parallel SRAP processing (PSRAP), was an example of this category. The final category, was that set of functions that were in the existing system, but which was implemented differently in the demonstration system, because of differences in either the hardware configuration, or the operating system. The data entry and display subsystem (DEDS) was an example of this category. The MPE services were replaced by combinations of commercial off the shelf (COTS) software, Multiple Virtual System (MVS), the Real-time Executive (RTX), and application services routines that made the executive services transparent to the application code.

4.3.2 Maintaining Traceability

Because the demonstration system's Tracking modules and algorithms were required to be functionally equivalent, and directly traceable to its ARTS IIIA system counterpart, the ARTS IIIA system Tracking modules were decomposed into the smallest possible, self-contained increments (tokens, modules). Smaller increments were much easier to analyze, design, and code, and therefore maintaining functional equivalency for these modules became much easier to implement.

Although the CDR Editor, Retrack, Interfacility, Keyboard, and the CDR Extractor are functions within the ARTS IIIA system, these respective functions were not directly traceable nor functionally equivalent within the demonstration system. Equivalency and traceability, are only evident in the converted code segments, algorithms, and capabilities that were retained from these functions. The CDR Extractor front-end filtering capabilities in the ARTS IIIA system are provided automatically in the demonstration system, eliminating or minimizing its functional equivalency. The CDR Editor, although not completely functionally equivalent, generates a hardcopy report that is functionally equivalent to the CDR Editor report generated by the ARTS IIIA system. The Retrack, Interfacility, and Keyboard functions, maintain some equivalency, but these functions were limited by the scope of this project, which was designed to recode only a subset of the current ARTS IIIA system.

Traceability was also maintained through the retention of the program or function names used in the ARTS IIIA system where possible, and through the unrestricted use of in-line documentation imbedded in the design and source code. ULTRA program names and labels were also used as Pascal/VS HOL program names and comments wherever possible. In some cases where the original ULTRA code was translated, existing labels were carried forward into the Pascal/VS code as in-line documentation (comments). It provided a convenient mechanism for locating the program, and/or sections of code within the program.

4.3.3 Traceability Matrix

This section describes the generation of the demonstration system's traceability matrix. The matrix was designed to map the ULTRA assembly lines of code to the demonstration system design and source code. The first phase of the mapping, was from ULTRA to PDL/Ada, and the second phase from PDL/Ada to Pascal/VS HOL.

The traceability matrix consisted of four sections. The first, and largest of these sections, entitled Tracking, was the combination of the Tracking (See Figure 16 Traceability Matrix Representation (Sorted NAS_MD and Section Numbers for Tracking)),Interfacility, and Keyboard modules. The last three sections were, the CDR Conversion, Retrack, and CDR Editor modules. All four sections were sorted on PDL/Ada, and on Pascal/VS HOL lines of code. Tracking, Retrack. and the CDR Editor, were also sorted on NAS-MD and section numbers. Retrack and the CDR Conversion modules had no NAS-MD documents for reference.

The sub-sections show the recode transition on a conceptual basis. Due to variations in the system's hardware and software architecture, and differences in the languages, and coding structure, the mapping varied from line by line to ranges of lines. This occurred where these ranges encompassed integral units, functions, algorithms, or design decisions, which the developer decided were irreducible conceptual entities.

The raw data for the Traceability Matrix was provided by the demonstration system software developers. Each developer filled in a blank traceability matrix panel with information specific to their area of expertise. The Tracking, Keyboard, and Interfacility panels were combined into one file entitled TRACKING. The three remaining panels, which are files unto themselves, were entitled CDR Conversion, Retrack, and the CDR Editor.

The basic process for acquiring the above information involved:

1) Obtaining and analyzing the NAS-MD's specific to the area of recoding.

2) Obtaining a compiled listing of the ULTRA code specific to the area of recoding.

3) Obtaining a copy of the Requirements Analysis Document.

4) Identifying functions in the ULTRA code which were to be converted as declared in the Requirement Analysis Document.

5) Converting the ULTRA functions to PDL/Ada and documenting the conversion in the traceability matrix.

6) Converting the PDL/Ada functions to HOL and documenting the conversion in the traceability matrix.

4.4 CDR Editor Listing Output Validation

The major constraint placed on the demonstration system output was the maintenance of functional equivalency of the CDR Editor output hardcopy report to the ARTS IIIA system CDR Editor output report.

To accomplish this the input data, the processing applied to the data, and the method of extracting the output had to be functionally and numerically equivalent.

The GFE CDR tape was converted to Pascal/VS format using an offline program. This program preserved each data item in the records the recoded system would process. The numerical format of some of the data was changed, some scaled integers were converted to real numbers, specifically in relation to the

Ultra Source Code		Program Design Language	High Order Language	NAS	Section	Description
IFI 01273-01295	I2	N/A	N/A	640	0500000000	Process Output Routing Field (IFOTROT)
IFI 01702-01836	I2	N/A	N/A	640	0500000000	Response Message Processing (IFACK)
IFI 00251-00321		N/A	N/A	640	0500000000	NY TRACON-ARTCC&ARTS Response Msg Xfer
COMA 00979-00981	P1	N/A	N/A	640	0602000000	Register save and lockout flag
COMA 00982-00989		TM111213 15800-16000	TM111213 00136-00148	640	0602000000	Check for discrete beacon code
COMA 00990-00993		TM111213 16100-16500	TM111213 00149-00160	640	0602000000	Set up while loops
COMA 00994-00996		TM111213 16600-16700	TM111213 00163-00165	640	0602000000	Check if comparing track to itself
COMA 00997-01000	P2	N/A	N/A	640	0602000000	Training track checks
COMA 01001-01017		TM111213 16800-18000	TM111213 00168-00192	640	0602000000	Check for matching beacon code
COMA 01018-01020	P1	N/A	N/A	640	0602000000	Restore registers
COMA 00437-00439	P1	N/A	N/A	640	0602010000	Set sr2 to temporary storage
COMA 00440-00445	P1	N/A	N/A	640	0602010000	Save temporary variables
COMA 00446-00463		TM111218 00132-00135	TM111218 00515-00523	640	0602010000	Save aircraft id & # characters in id
COMA 00464-00465	P2	N/A	N/A	640	0602010000	Training tracks
COMA 00466-00466		TM111218 00136-00137	TM111218 00524-00527	640	0602010000	Call DUPIDC to get duplicate acid's
COMA 00467-00472	P1	N/A	N/A	640	0602010000	Restore temporary variables
IFI 00322-00821	I1	N/A	N/A	640	1000000000	Discarded Messages
IFI 00322-00423	I1	N/A	N/A	640	1000000000	Acceptance Message (DA)
IFI 00424-00467	I1	N/A	N/A	640	1000000000	Retransmit (DR) and Rejection (DX) Msg
IFI 00468-00493	I1	N/A	N/A	640	1000000000	Data Test Message (DT)
IFI 00494-00526	I1	N/A	N/A	640	1000000000	Test Data Transfer (TR)
IFI 00527-00607	I1	N/A	N/A	640	1000000000	Track Update Message (TU)
IFI 00608-00718	I1	N/A	N/A	640	1000000000	Initiate Track Messge (TI)
IFI 00719-00821	I1	N/A	N/A	640	1000000000	Accept Transfer Message (TA)
CDR 02195-02220	P1	N/A	N/A	648	0302010400	Eliminate filter check and buffer manip
CDR 02221-02287		TM111240 12300-16600	TM111240 00350-00402	648	0302010400	Construct output msg for cdr extractor
CDR 02283-02320	P1	N/A	N/A	648	0302010400	Delete cts msg and buffer manipulation
CDR 01936-01965	P2	N/A	N/A	648	0302010600	Delete etg,subsyst,ass status filt chks
CDR 01967-02002		TM111204 11100-13200	TM111204 00465-00508	648	0302010600	Construct output mssge for cdr extract
CDR 02003-02005	P1	N/A	N/A	648	0302010600	Eliminate user count
CDR 02006-02012	P1	N/A	N/A	648	0302010600	Register manipulations
CDR 02881-02887		TM111205 12000-12160	TM111205 00266-00269	648	0302010600	Get trk num, check if associated
CDR 02888-02890		TM111205 12200-12400	TM111205 00270-00273	648	0302010600	Check to see if trk is ARTCC controlled
CDR 02891-02895		TM111205 12900-13600	TM111205 00277-00283	648	0302010600	Get contrlr symbol from symt table
CDR 02896-02901		TM111205 12600-12700	TM111205 00275-00276	648	0302010600	Get catb words and update contrl. symbl
CDR 01529-01537	P4	TM111206 08900-09400	TM111206 00127-00132	648	0302010600	Compute time of day using system funct.
CDR 02865-02868		TM111241 10900-11000	TM111241 00192-00196	648	0302010600	Determine index to SYMT table using cts
CDR 02869-02872		TM111241 11100-11300	TM111241 00197-00199	648	0302010600	Get the associated kb subset from symt

APPENDIX B.4.1 Footnotes: Sorted NAS-MD and Section Numbers for Tracking

P1 - For use in assembly language only; not used in HOL (example : linking, loading, saving registers)
P2 - Function is not part of requirements document (requirements document says no)
P3 - Not necessary - Different architecture (example : Multiprocessor to single processor)
P4 - Different implementation in PASCAL (example : Ultra replaced by a built_in function; Equivalent function different)
P5 - Other (explain in the comment section when code 05 is used)

I1 - These messages are discarded, therefore the Ultra code is not implemented for this demonstration.
I2 - The function is not implemented for the demonstration, it is listed for continuity of ULTRA sequence numbers.
I3 - Library TR0501.DIFI.PDL
I4 - Library TR0501.DTRACK.PDL
I5 - Library TR0502.DIFI.PASCAL
I6 - Library TR0503.DIFI.PASCAL

Figure 16 Sample from Traceability Matrix velocity, and x and y coordinates. This new format (or type) was perpetuated throughout the entire system resulting in improved precision.

The demonstration system team performed two distinct types of comparison to verify the functional equivalence of the two CDR Editor outputs. One method required the input of filters to select specific information (e.g. Beacon codes, sensors, azimuth). The generated CDR Editor output was then compared to the ARTS IIIA CDR Editor output to prove the correlation exists. The first, entry on the demonstration system output was then located, and the system time, range, azimuth, beacon code and sensor was then verified for functional equivalency to the ARTS IIIA CDR Editor respective fields.

The alternate method of verifying the functional equivalency of the demonstration system CDR Editor output to the ARTS IIIA CDR Editor output entailed inputting a data class only, with no filter selection. The output from this method was expected to yield the identical information as the ARTS IIIA CDR Editor. This was verified by examining the demonstration system generated CDR Editor report, which was done in the same manner as the verification of the first method.

The demonstration team and the FAA also compared the outputs from the CDR Editor, generated from runs of the New York TRACON system and the recoded system, to verify functional equivalence. To facilitate the job of comparing the outputs the team took several steps prior to the validation, they included:

- o Recoded the CDR Editor, so that it would be functionally equivalent.
- o Designed the CDR Editor output listing to look exactly like the FAA-furnished output.
- o Wrote CDR analysis programs to simplify analysis of the CDR input file.

Comparisons between the operational and recode output provided a quick and objective method of accurately identifying differences. Each difference was analyzed to distinguish between software errors and architectural differences. Architectural differences were due primarily to the use of a different processor which ran at a different speed and different numerical representations. Slight differences in timing and numerical accuracy were expected.

The following discrepancies were noted:

- The ranges and x,y-values had .01 discrepancy at times due to use of floating point rather than scaled integer notation.

- The ranges had discrepancies due to the use of the square root function of x**2 + y**2 as compared with the approximation method used in the GFE system.

- Some differences in time were noted due to use of 1/1024 seconds in the NY TRACON system versus 1/1000 in the demonstration system.

- The demonstration system had fabricated keyboard messages.

- The only interfacility messages that were converted were the FP, AM, and CX messages.

- When comparing the TD messages, a flight had to be found and followed.

5.0 Project Management

The information contained in this section describes the tools, methods, guidelines, and standards used to manage the project.

A single set of program controls, based on government standards, was developed to monitor and control this project. These controls were tailored specifically to a project of short duration. Their procedures and standards were documented in the Program Management Plan (CDRL A001). Specific measurement points and milestones were assigned to verify the quality of the products produced and any cost or schedule variance. The team's definition and use of a single set of software performance controls for software deliverables (CDRL A001 - Program Management Plan : Section 3.2.2.1 Performance Control), internal products, and program status documents, provided the capability to measure and compare adherence to the program controls for each deliverable.

The master milestone schedule was documented in the technical proposal. Detailed development milestones were documented in the software development plan. These detailed milestones were monitored for adherence to overall project schedule.

The project work breakdown structure (WBS), based on the software system life cycle activities, enabled management to track cost/schedule performance for the program and identify and/or forecast possible problem areas, bottlenecks and dependencies. Initial source lines of code (SLOC), estimates by computer program configuration item (CPCI), were presented in the technical proposal. They provided management with a cross-reference for each program product developed or converted.

The requirements analysis specified the detailed technical activity necessary to implement the project objective. Project requirements were documented and baselined in the Requirements Analysis Document (CDRL A002). This document was divided into two sections. The first, the functional specification, was organized to correspond to the existing system functional definition (NAS MD). The second described the system and architectural changes that were not present in the current operational system. This document was used as the basis for the design and project verification activities.

Standards and controls for the design, implementation and test activities are documented in the Program Management Plan. Design and code inspections by technical peer groups were conducted to ensure and enforce project standards and to verify the technical correctness of the product. Data management was established through the use of the software development laboratory (SDL), which was used for storing disk data sets and magnetic tapes; and through the software library, which was used for storing pertinent documents. Configuration control was maintained by the establishment of two separate levels of product control: a baseline level and a development level. Risk analysis identified and categorized technical risks and defined a risk reduction approach.

6.0 Statistical Summary

This section provides a statistical summary of the results of the system's development including total lines of code for each phase of the contract (PDL/Ada, ULTRA, Pascal/VS) design issues, and a PTR analysis. Also included in this section is information pertaining to a poll of each individual developer's approach to the reengineering process, a graph illustrating the results of questionnaires and the raw data collected from this process.

6.1 System Development Results

The demonstration system's team developed three hundred forty (340) Pascal/VS procedures. One hundred fifty six (156) procedures, mostly tracking, were implemented with no functional deviations from the original ULTRA implementation. The remaining procedures consisted of newly developed support code and modifications to the current ARTS IIIA system.

These Pascal/VS procedures made up the demonstration software and were equivalent to fifty-three thousand (53,000) lines of ULTRA source code. This was converted into forty-seven thousand lines (47,000) of high and low level PDL/Ada, including commentary. The PDL/Ada design was implemented in eighty-three thousand (83,000) lines of commented Pascal/VS source code (approximately 32,000 lines without commentary). This source code ratio, 53,000 lines of ULTRA to 32,000 lines of Pascal/VS, suggests a workload estimation factor in the vicinity of 1.6 ULTRA to one Pascal/VS source line. This metric will prove useful in estimating other program conversions.

The Demonstration System's team identified and documented fifty eight (58) DIs during the conversion and redevelopment effort. Each of the 58 DI's was resolved within a week, supporting the perception that the design methodology was working as expected.

The Demonstration System's team wrote seventy-eight (78) Program Trouble Reports (PTRs) during the project. In view of the scope and complexity of the demonstration system, the PTR count is low. Of these 78 PTR's, 56 were issued during development testing, 18 during integration testing, and 4 were issued as a result of demonstration testing.

The PTR's are summarized in the following table:

PTR Type	DEV	INTG	DEMO	TOTAL
Improvements	21	9	1	31
Errors	35	9	3	47
TOTALS	56	18	4	78

Errors corrected during unit testing and initial build testing were not tracked and exhaustive system tests were not performed. We experienced an error rate of 1.5 per one thousand SLOC.

The demonstration system was not thoroughly tested and errors still remain which may contributes to the low error rate. Anomalies in the tracking output also exist. These anomalies, although existent within the tracking output, are not indications of errors in the tracking algorithms. Based upon the limited duration of this effort (9 months), further testing and investigation may yield minimal additional errors.

6.2 Methodolcgy Questionnaire Poll

A poll of each developer of the demonstration system was conducted to capture relevant information not depicted by any reporting activity, or deliverable. To ensure objectivity the poll was conducted in private on a one to one basis, (i.e interviewer to developer), during the post-development period of Build 3 but prior to it's implementation.

Defining each individual developer's personal approach to the conversion process was the purpose of the Methodology Questionnaire. It permitted developers to describe different aspects of the conversion process; including, module criticality, level of conversion, algorithmic complexity, and code organization. Developer responses to these questions and analysis of the data provided important information to the Methodology Document.

Perhaps the most important result of the questionnaire analysis was a graphic illustration of the correlation between algorithmic complexity in the original ULTRA source code and the number of PDL lines of code. This analysis showed that the more complex and mission critical the module is, the greater the number of PDL lines of code produced in the conversion process. An outline illustrating the format followed in the questionnaire analysis and the graphs plotting PDL/ULTRA SLOC ratio, which were derived from the responses to the questionaire poll, are included in Appendix C.

Appendix A. System Parts and Their Work

The information contained in this Appendix is an excerpt of System Parts and their Work (Expected Behavior of the Software) which was initially presented in the Program Management Plan CDRL A001.

- o There will be six classes of level-1 packages:

  - oo Monitor (M)
  - oo Offline (O)
  - oo Control (C)
  - oo Interactive (I)
  - oo Pipeline (P)
  - oo Data (D)

- o The Initialization and Termination package (M) will synchronize the system startup and shutdown by sending and receiving notification software messages to and from the other packages. It will process operator requests to start and stop the job. (In operational mode there would be an interactive interface with the operator; in test mode, the interaction with this package.) If the run is terminated gracefully for other reasons, such as a processing timing parameter is exceeded, this package will be invoked. Its retained data will included the names of the other packages and information about their processing states.

- o Messages Control (M) will field the communications primitives issued by the other packages and interface with RTX to provide the appropriate (time and space) resources. Its retained data will include the names of the other packages and information about their communications states. It will error check messages at the link level to ensure that message types and destinations are correct and will terminate processing if an error is found.

- o Timing Control (M) will periodically determine if the pipeline deadlines are being met; if they are not it will terminate processing. Its retained data will include the critical system events and expected elapsed times for each. It will terminate processing if a critical event does not occur within the expected time. It will provide system time services to the other packages.

- The offline package, CDR Editor (O), will execute in batch mode under MVS. It will use MVS services directly to read the CDR (output) tape and generate a listing of the online system's journal. The listing will show that the demonstration system functions are equivalent to those in the current New York TRACON system.

- The online application packages and their primary work units are:

DED Access (C)	== Software messages
Retrack (C)	== CDR records
CDR Extraction (C)	== CDR Records
Interfacility (I)	== Flights
Keyboard (I)	== Controller commands
Target Acquisition (P)	== TRACON airspace
Tracking (P)	== TRACON airspace
Display (P)	== Controller sector (a set of tracks and targets within a sensor)
Common Updatable (D) Data	== Software Message

- The DEDS Access package provides -- through commercial off-the-shelf and developed software -- the link level I/O support between the Display Outputs application and the DEDS. Because controller commands (keyboard inputs) are input from Retrack only, DEDS Access supports outputs only. DEDS Access uses MVS services to provide channel and interrupt level I/O support. It retains data about the DACU and the display generator protocol.

- Retrack reads the CDR tape containing the recorded transactions from a previous execution of the full ARTS system (not our demonstration system). Retrack reads target, controller command and flight records into its internal buffers and passes the target records to the Target Acquisition package, the commands to Keyboard, and the flight data to Interfacility. Retrack does not pass work that has already been identified on the CDR tape as in error. Retrack sends second-order messages, modifications to existing flights and tracks, to Interfacility and Keyboard. If the messages are out of sequence they are recorded as errors on the CDR output by Keyboard or Interfacility. (Retrack does not interface directly with the CTS as in the current New York TRACON system.)

- CDR Extraction receives software messages from the other online packages, transforms them to CDR records and writes the records to the CDR (output) file using a standard MVS access method.

DATA ELEMENT DICTIONARY

The Data Element Dictionary (DED) contains all the tracking subsystem global data element names and descriptions. The main purpose of this DED is to bring the data element specification control under Configuration Management (CM). This central control of the data element convention, data typing, database assignment, and description will eliminate confusion among the software development personnel during system PDL, code, and integration testing. The configuration manager will be the only person allowed to modify the file copy of the DED.

In arriving at the new DE names, you will note that the old DE names map from a single old name to multiple new names. The reason for this is as follows: In the old DE list, the old names were associated with UNIVAC 30 bit words and arrays (tables) of words. The fields identified at the sub-word or bit level are not given a discrete DE name; however, in the new DE, all the fields that appear at the sub-word level have been assigned a discrete DE name. Since multiple fields (sub-words) exist for single words in the old DE, correspondingly, multiple new DE names exist for single old DE names. This assignment of new, descriptive DE names for all defined fields in the database will enhance the readability, reliability, and maintainability of the software. The following section describes how Higher Order Language data structures can support the DE typing to the bit level with no problem.

The DE will be maintained using the DBASE III+ software package. The structure of each DE record is as follows:

Field	Field Name	Field Description
1	COMPANY	company respon. (D - DTC, I - IBM, P - PJA)
2	DATANAME	Old data element name (existing system)
3	DATABASE	Old database assignment
4	PAGENUMBR	Page number in database doc. section
5	P1	A referencing procedure name
6	S1	An indicator specifying whether the referencing procedure sets or uses the data element
7 - 24		Fields 4 and 5 are repeated for nine more procedures
25	TYPE	New data element type code
26	VARNAME	New data element name
27	DBNAME	New database name
28	DESCRIPT	Data element description

Notes: 1) The set/use indicator is defined as follows:
- 1 - set by referencing procedure
- 2 - used by referencing procedure
- 3 - both set and used by referencing procedure

2) The new data element type code is defined as follows:

S - character string
C - character
L - boolean
i - integer (short)
I - integer (long)
r - real (short)
R - real (long)
B - Bit
A - array (table)
P - pointer (address)
E - enumerated type

Appendix B-2. Description of the Mapping Tables

SITE & SYSTEM DATA DICTIONARY ELEMENT (DED)

The Site and System DED includes all the variables in 3.5.2 of Coding Specification and sections 2 to 3 of NAS-MD 643. This DED is created by the combination of variables in the files CSITEQ, DSITEQ, MSITEQ, TSITEQ, SYSEQ0 and TI of TRACON.A504.DATA. Any duplicated elements in CTS have been discarded. If the element is only referenced by system database (SDB1, SDB1RO, SDB2, DBASEC, DBASED and DBASEE), it would be located in mapping table. The structure of the mapping table follows:

DATANAME	data element name (for the existing NY TRACON).
SECTION	section number appears in NAS-MD-643.
PAGENUM	page number appears in NAS-MD-643.
DATABASE	database name from TRACON.A504.DATA in existing NY TRACON
SDB1	element or table name which uses the DATANAME in SDB1
SDB1RO	element or table name which uses the DATANAME in SDB1RO
SDB2	element or table name which uses the DATANAME in SDB2
CFGT	element or table name which uses the DATANAME in CFGT (DBASEE).
CDRD	element or table name which uses the DATANAME in CDRD (DBASEC)
SUBS	element or table name which uses the DATANAME in SUBS (DBASEC)
MBUF	element or table name which uses the DATANAME in MBUF (DBASED)

Appendix C Description of Questionnaire Analysis

1.0 Interview Questionnaires

The interview questionnaire was designed to obtain two distinct classes of information regarding the methodology process:

> Numerical assessments of the decisions made during the recoding process to be used to mathematically relate these factors to the accomplishment of the project objectives. The data from this information set is reported, discussed and analyzed.
>
> Objective assessments of the methodology originally intended, problems encountered, and the modifications made during actual use to correct them, as well as any suggestions that would improve the process for use on future projects. The overall impressions and common concerns of the developers in their discussions of this second class of information is the purpose of this supplement.

The interview questionnaire was intended to capture the reservations, recommendations, concurrences and differences of opinion, and approaches of the developers of the demonstration system. It was a concrete focus for the ideas to be discussed during the interviews and served as a record of their impressions.

2.0 The Interview Process

Each developer on the project with responsibility for one or more distinct NAS-MD functions was given a questionnaire form. Each was given a copy of the classification descriptions and values. The developers completed the numerical evaluation and initial comments following the completion of the coding phase of the project. The interviews were scheduled during the system integration phase. No attempt was made to guide the contents of the comments other than the general instructions that they were an opportunity to report difficulties and make recommendations for future recoding projects. The actual interviews were conducted in groups of two or three developers from one of the team companies, and one or two interviewers. In order to ensure maximum freedom of expression, and confine the discussions to a small number of issues, technical, supervisory and management personnel were all interviewed separately.

The interview was divided into three major sections, each covering a specific portion of the project development. In section one, developers were asked to discuss their overall approach to the project. Differences between the New York TRACON and a more "traditional" project were addressed in section two. The third section, tools, discussed what tools were, or would have been, especially helpful in completing this project.

3.0 Summary of Questionnaire Discussions

The methodology questionnaire permitted developers to describe different aspects of the conversion process; including, module criticality, level of conversion, algorithmic complexity, and code organization. Developer responses to these questions, and analysis of the data, provided information used in preparation of the Translation and Verification Methodology Document. The areas of discussion and comments have been loosely grouped for reporting purposes.

3.1 Training and Orientation

As the developers considered their initial project approach, many felt an orientation period would have been beneficial. This ranged from a simple orientation lecture to an intensive two-week training period.

The majority of the developers spoke about the difficulty in reading the original ULTRA source code. A training session in ULTRA, including architecture, would have provided valuable assistance. It was generally felt that more time should have been spent with the entire team understanding ULTRA and its environment.

It was also suggested that training in Pascal be included. The developers suggested that even individuals who knew Pascal be required to attend such a training session. The Pascal used for this project was not standard, and it was the consensus that all developers should begin the project with the same information.

Developers also requested training in CMS/MVS. They believed they were not able to gain maximum benefit from a powerful system due to a lack of both documentation and knowledge of the system.

A final training suggestion addressed the Air Traffic Control System as a whole. Some developers felt knowing the "big picture" would have helped them understand their role in the New York TRACON Project. Suggestions included a field trip to a TRACON, a field trip to the FAA Technical Center in N.J., and/or an orientation lecture regarding FAA policies and procedures.

3.2 Analysis of Requirements Phase

Requirements Analysis was the second major subject of discussion in the interview. It was unanimously agreed that this stage was one of the most important. One individual stated that "A lot of work early is worth a little work later." It was also said that "Errors encountered at test are more costly than those encountered during requirements analysis."

In contrast, because this project did not follow a standard/classical software cycle with which they were familiar, it was difficult to know what to expect. For that reason, many developers felt too much time was spent "designing the design." This caused problems later on when time constraints interfered with testing.

A positive result of the requirements analysis stage was the development of a data element dictionary. This dictionary was widely praised by all developers. It was agreed that the development of this dictionary should be the one of the first tasks completed in another re-coding project. One developer stated that "if we hadn't done an analysis of the data base, we wouldn't have gotten this far."

3.3 Coding and Testing

As the developers discussed coding, it was the prevailing opinion that reading the original ULTRA source code to determine requirements was also important. Opinions on this subject differed as to how detailed this stage of the conversion should be. Some developers, such as those working on DISPLAY and CDR CONVERSION, were not affected because their sections were generally re-designed. However, those who performed a straight re-code felt a line-by-line analysis of their section was required.

3.4 Comparison With Classical Projects

Developer Interviews also discussed the differences between a recoding project and a "traditional" software project, one that involved primarily creative programming. Generally, most developers felt differences were not pronounced.

The biggest difference noted was that a traditional software cycle was not followed; therefore, one did not know what to expect.

Many developers felt a recoding project would never be as difficult again because they were "leaving a trail behind them." All developers expressed a real sense of responsibility and made sure clear documentation was kept.

The importance of teamwork on a recoding project was also stressed. Developers believed teamwork was more important on this project than on previous projects with which they had been involved.

3.5 Development Tools

The tools used in the Demonstration System project were generally felt to have been helpful.

The ULTRA source code comments were mentioned most often as providing the best assistance. Some developers wished these comments were more detailed; but overall, they were invaluable.

The coding specs (CPFS) also provided an excellent overview; but again, were not always accurate or detailed enough.

The NAS-MD's were an important information source. A minor complaint was that the documents were not always up-to-date.

The data element dictionary, as discussed in section one, was roundly approved as an important design, development, and testing tool.

There were tools that were not available to the developers, but would have been helpful. These included a more user-friendly editor, an automatic formatter, and a programmer's notebook.

3.6 Suggestions

Many suggestions for similar projects were given. These include:

A. Require internal (peer) reviews of the PDL.

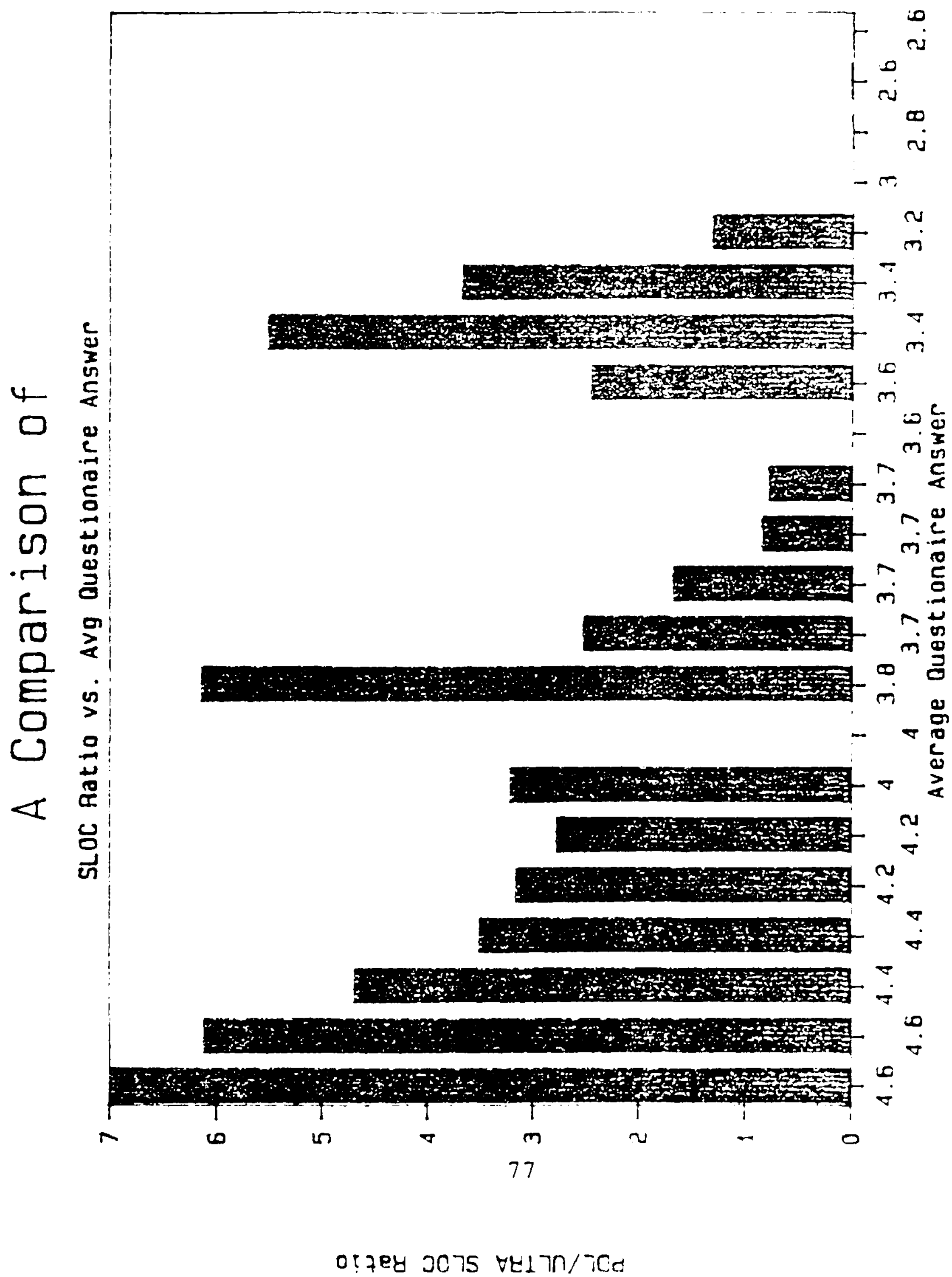
B. Different developers should code from ULTRA to PDL/Ada and from PDL/Ada to Pascal. This would catch many problems before test.

C. Provide an expert or experts to answer questions. This should be someone who knew ULTRA, was familiar with FAA policies and procedures, and could apply what was being done to the 'real world.' All developers expressed appreciation to the controller who became available toward the end of the project and to the assistance of the FAA SEI representative.

D. Specific background and experience was considered beneficial by many project developers. These included a physical science background, assembly language experience of any kind, a mathematical background, real-time programming experience, and high-level programming experience.

4. Analysis of Questionnaire Data

Perhaps the most important result of the questionnaire analysis was a graphic illustration of the correlation between algorithmic complexity in the original ULTRA source code and the number of PDL lines of code. This analysis showed that the more complex and mission critical the module is, the greater the number of PDL lines of code produced in the conversion process. Following is a table illustrating the format following in the questionnaire analysis. Graphs plotting PDL/ULTRA SLOC ratio are also included.


A Comparison of
SLOC Ratio vs. Avg Questionaire Answer
PDL/ULTRA SLOC Ratio
Average Questionaire Answer
0
1
2
3
4
5
6
7
4.6
4.6
4.4
4.4
4.2
4.2
4
4
3.8
3.7
3.7
3.7
3.7
3.6
3.6
3.4
3.4
3.2
3
2.8
2.6
2.6
77

APPENDIX C
DEVELOPER INTERVIEWS
OUTLINE

Interviews were divided into three sections. These were:

A. Project Development
(Conversion Process)

B. Differences between this and a traditional
software project

C. Tools

A. Project Development

1. Provide an initial overall introduction/training period
for project team members.
Training period should include:
    * ULTRA (architecture)
    * PDL/Ada
    * System (MVS/PTX)
    * ATC system overview (FAA policies & procedures)

2. Requirements Analysis
    * More time should be spent on this stage. "A lot of work early is worth a little work later."
    * Data Element Dictionary
    * "Errors encountered at test are more costly than those encountered during requirements analysis."

3. Return to ULTRA code to determine coding requirements.
    * Some developers were not affected by this (e.g. DISPLAY).
    * Done at different levels by different developers.

4. Time Frame
    * Not realistic
    * Led to difficulties with integration

Suggestions for Project Development:

1. Require internal PDL reviews (peer reviews)
2. Different developers should code from ULTRA to PDL/Ada and from PDL/Ada to Pascal.
3. Make sure an expert is available to answer questions.

5. Developer Experience. The following were cited by developers as skills/experience they found especially helpful in the completion of this project.
   * Physical science background
   * Assembly language experience
   * Mathematics background
   * Real-time programming experience
   * High-level language programming experience

B. Differences between this and a more traditional software project.

1. Differences were not pronounced.
2. Teamwork was very important.

C. Tools

1. Generally very helpful.

2. Comments in ULTRA source code were very helpful but could have been more detailed.

3. Coding Specs (CPFS)
   Good overview, but not always accurate.

4. NAS-MDs
   Very helpful, but at too high a level. Not updated.

5. Data Element Dictionary

Tools Missing:

1. More detailed and complete ULTRA comments.
2. More user-friendly editor
3. Automatic formatter
4. Programmer's notebook

Definitions for Questionnaire

I) Criticality: A measure of the system's dependancy upon the software module in order to accomplish the missions objective. The more the system depends on the module, the more mission critical is the module. The less the system depends on the module the less mission critical is the module.

How do you rate criticality of this module?

4) Mission Critical

3) Somewhat Mission Critical

2) Somewhat Not Mission Critical

1) Less Mission Critical

II) Conversion level: The level of precision with which the original source code is translated. The more detailed the replication, the higher the conversion level. The less detailed the replication, the lower the conversion level.

What Conversion Level did you use to recode this module?

6) Translate (Line by line)

5) Recode (Software thoughts)

4) Rewrite (Change in original flowchart)

3) Redesign (Modular design changes)

2) Replace (Requirements modified)

1) Discard/Add (Entire module is added/discarded)

III) Algorithmic complexity: A measure of the complexity of the computational method. The more difficult it is to understand the logic, the more complex the computational method. The less difficult it is to understand the logic the less complex the computational method.

How do you rate the Algorithmic complexity of this module?

4) Complex

3) Somewhat Complex

2) Somewhat Not Complex

1) Not Complex

IV) Organization of the code: A measure of varience between the physical and logical flow of code. The more the varience, the more unstructured the code. The less the varience, the less unst.uctured the code.

How do you rate the Organization of the code in this module?

4) Unstructured

3) Somewhat unstructured

2) Somewhat structured

1) Structured

V) Tokenization: The act of dividing the source software code into smaller more manageable software units. The higher the level of tokenization the higher the replication of the original code, the lower the level of tokenization the lower the replication of the original code.

What level of tokenization did you choose for this module?

6) Identity (Line by line)

5) Lexical (Source lines equal to a flowchart symbol)

4) Logical (Predicates, loops, and linear)

3) Functional (Subroutine calls)

2) Process (Routine)

1) Modular (Entire Module)

DUE
10/9/87
Thur

METHODOLOGY INTERVIEW

Developer's Name: BOSWORTH

Developer's Supervisor: EARLY

Date: 4/8/87

Company: STC

Assembly Unit Name: RETRACK

Module* Name	Question/Answer I.	II.	III.	IV.	V.	ULTRA SLOC	PDL SLOC
PROCESS_IFY_MSG TMØ5ØGØØ	N/A	N/A	N/A	N/A	N/A	Ø	84
IFY_APP TMØ5Ø6Ø2						Ø	51
CHR_CHK TMØ5Ø6Ø3						Ø	141
FP_CHK TMØ5Ø6Ø4						Ø	59
DA_CHK TMØ5Ø6Ø5						Ø	35
AM_CHK TMØ5Ø6Ø6						Ø	73
CX_CHK TMØ5Ø6Ø7	N/A	N/A	N/A	N/A	N/A	Ø	34
FP_PROC TMØ5Ø62Ø	4	5	3	3		FPINP 22	56
DA_PROC TMØ5Ø63Ø	4	5	3	3		DAOUT, FINDMATCH 50	85
AM_PROC TMØ5Ø64Ø	4	5	3	3		50	43
CX_PROC TMØ5Ø645	4	5	3	3		50	40

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: Marie Brown

Developer's Supervisor: ____________

Date: 4/8/87

Company: DTC

Assembly Unit Name: Editor

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
Editor	I. 2 II. 3 III. 2 IV. 2 V. 2	10300	2300
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: Dominic Cheung

Developer's Supervisor: ____________

Date: 4/8/87

Company: DTC

Assembly Unit Name: ____________

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
CONVERSION	I. 4 II. 1 III. N/A IV. N/A V. N/A	N/A	2000
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

Since conversion program do not have the counterpart in the Ultra code. A lot of the ~~question~~ answers ~~is~~ are N/A.

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: HOSSEIN & EARLY

Developer's Supervisor: EARLY

Date: ______

Company: ______

Assembly Unit: [illegible]

String Leader: ______

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
PROCESS_[illegible] TM050200	I. N/A II. N/A III. N/A IV. N/A V. N/A	6	74
BUILD_FLIGHT-DATA TM050[illegible]	I. N/A II. N/A III. N/A IV. N/A V. N/A	0	82
PROCESS_SEA-TAB TM05[illegible]	I. N/A II. N/A III. N/A IV. N/A V. N/A	0	59
ACID_CHECK TM050203	I. 2 II. 2 III. 2 IV. 3 V. 5	1977-1989	34
SET PREVIEW TM050204	I. N/A II. N/A III. N/A IV. N/A V. N/A	0	57
AUGMENT-DISCR-BEAC.TAB TM050801	I. N/A II. N/A III. N/A IV. N/A V. N/A	0	45
DEL-DISCR-BEAC_TAB TM050802	I. N/A II. N/A III. N/A IV. N/A V. N/A	0	50
SEARCH-DISCR-BEAC-TAB TM050803	I. N/A II. N/A III. N/A IV. N/A V. N/A	0	38
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within an assembly unit. Each assembly unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: ______

Developer's Supervisor: ______

Date: ______

Company: ______

Assembly Unit Name: RETRACK

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
PROCESS_SENSOR-MSG TM[illegible]	I. 4 II. 5 III. 3 IV. 1 V. 2	0	97
SAVE_TARG-REPT_MSG TMOSO402	I. 4 II. 5 III. 1 IV. 1 V. 2	1110-1200	12
SAVE.RADAR-ONLY MSG TMOSO403	I. 4 II. 5 III. 1 IV. 1 V. 2	1110-1200	[illegible]
SAVE_SECTOR-TIM-MSG TMOSO401	I. N/A II. N/A III. N/A IV. N/A V. N/A	Ø	12
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		

A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: M L Ichniowski

Developer's Supervisor: C F Hayes

Date: 4/9/87

Company: IBM

Assembly Unit Name: UDREXT

Module* Name	Question/Answer					ULTRA SLOC	PDL SLOC	
INITSEG	I. 3	II. 4	III. 2	IV. 1	V. 3			[illegible]
	I.	II.	III.	IV.	V.			
RECDSEG	I. 3	II. 3	III. 2	IV. 1	V. 3		[illegible]	[illegible]
	I.	II.	III.	IV.	V.			
TERMSEG	I. 3	II. 4	III. 2	IV. 1	V. 3		[illegible]	[illegible]
	I.	II.	III.	IV.	V.			
	I.	II.	III.	IV.	V.			
	I.	II.	III.	IV.	V.			
	I.	II.	III.	IV.	V.			
	I.	II.	III.	IV.	V.			

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: Margaret McCandless

Developer's Supervisor: Connie Hayes

Date: 4/22

Company: IBM

Assembly Unit Name: DATA

Module* Name	Question/Answer	ULTRA SLOC	PDL SLCC
	I. 4 II. 3/4 III. 3 IV. 3 V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

System & Site Data Constants

METHODOLOGY INTERVIEW

Developer's Name: L. MASS[illegible]

Developer's Supervisor: C. [illegible]

Date: 4/5/87

Company: I[illegible]M

Assembly Unit Name: TT[illegible]

Module* Name	Question/Answer					ULTRA SLOC	PDL SLOC
	I.	II.	III.	IV.	V.		
CATS[illegible]	4	1	3	1	–	–	
TM[illegible]38	4	1	3	1	–	–	
[illegible]	4	1	3	1	–	–	

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: J.R. ATKINSON

Developer's Supervisor: Tom BAXTER

Date: ________

Company: PJA

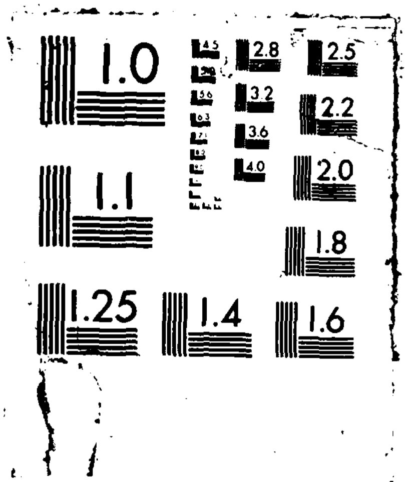
Assembly Unit Name: TRACKING

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
TPUR	I. 4 II. 5 III. 4 IV. 4 V. 5	913	4300
TROUT	I. 5 II. 3 III. 2 IV. 4 V. 5	182	672
SLINK	I. 1 II. 5 III. 3 IV. 4 V. 5	276	954
MAT	I. 1 II. 5 III. 3 IV. 4 V. 5	211	672
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within [illegible] unit has one or more distinct modules.

AD-A189 546 NEW YORK TRACON DEMONSTRATION OF PROGRAM RECODING SOFTWARE TRANSLATION AN..(U) FEDERAL AVIATION ADMINISTRATION TECHNICAL CENTER ATLANTIC CIT.. AUG 87 2/2

UNCLASSIFIED DOT/FAA/CT-87/33 DTFA03-85-C-0058 F/G 12/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

METHODOLOGY INTERVIEW

Developer's Name: LALITHA BHAT

Developer's Supervisor: TOM BAXTER

Date: ________

Company: PJA

Assembly Unit Name: TRACKING

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
TEXEC	I. 4 II. 3 III. 3 IV. 3 V. 4	220	1215
TPRED	I. 4 II. 5 III. 4 IV. 4 V. 5	428	1386
TCRSS	I. 4 II. 6 III. 4 IV. 4 V. 5	179	1251
TPSEC	I. 4 II. 6 III. 4 IV. 4 V. 5	667	4095
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		
	I. __ II. __ III. __ IV. __ V. __		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: CLARKE THOMASON

Developer's Supervisor: TOM BAXTER

Date: ________

Company: PJA

Assembly Unit Name: PSRAP

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC w/comments
PSRAP	I. 4 II. 3 III. 3 IV. 3 V. 3	1450	1935
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: CLARKE Thomason

Developer's Supervisor: TOM BAXTER

Date: ________

Company: PJA

Assembly Unit Name: TRACKING

Module* Name	Question/Answer	ULTRA SLOC	PDL w/comments SLOC
TUD	I. 4 II. 5 III. 2 IV. 3 V. 5	450	2774
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: Jeff Marcus

Developer's Supervisor: Tom Baxter

Date: 4/7/87

Company: PJA

Assembly Unit Name: Tracking

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
TEDC	I. 4 II. 6 III. 3 IV. 3 V. 5	172	[illegible]
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

METHODOLOGY INTERVIEW

Developer's Name: MICHAEL GANDEE

Developer's Supervisor: TOM BAXTER

Date: ________

Company: PJA

Assembly Unit Name: TRACKING

Module* Name	Question/Answer	ULTRA SLOC	PDL SLOC
TINIT	I. 4 II. 5 III. 4 IV. 4 V. 5	524	1849
Common Subroutines	I. 4 II. 5 III. 3 IV. 3 V. 5		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		
	I.__ II.__ III.__ IV.__ V.__		

* A module is a distinct software unit within an assembly unit. Each unit has one or more distinct modules.

Appendix D. Glossary of Terms

The following represents a glossary of terms used in the development of the N.Y. Tracon Demonstration of Program Recoding. Emphasis has been placed on the retention of the terminology and content used in the N.Y. Tracon A5.04 System wherever possible.

ABSTRACT DATA The specification of a data type and a "hidden" TYPE (from its user) body defining a more concrete representation.

ACCESS TIME The time interval between the instant information is requested from memory and the instant the information is available.

ACKNOWLEDGE The indication of the status of data on the input/output lines.

ACTIVE STATUS Refers to a track which is being correlated, corrected and predicted every scan. A track which is not in flight plan status, store status, suspend status or tabular coast status.

ACTIVE COAST Refers to an active track which has failed to STATUS correlate but which has not met the conditions for tabular coast status. A track in active coast status has its next scan position predicted.

ACTIVE HANDOFF An active coast track in handoff to another STATUS controller (N.Y. TRACON or ARTCC).

ADA PACKAGE A software unit that has a specification part, a body part and a procedures part that defines an abstract data type and behaves as a state machine.

ADAPTATION Unique site dependent data required by the operational program to provide the flexible capability necessary to allow it to function at individual sites.

ALARM Refers to a signal or indicator which warns of a abnormal or out-of-tolerance condition.

ALPHANUMERIC CHARACTER A letter or number.

APPLICATIONS The ARTS work exclusive of operations, tracking for example; the software that automates that work.

APPLICATION MESSAGES Units of data by which the system and the real (air traffic) world communicate, beacon and radar targets, tracks and flights.

APPLICATION Work units, containing system application WORK messages, that define the processing constraints of the software; a process must complete one system application work unit, such as a track, before starting the next.

APPLICATION WORK	An integral input from the system application STATION COMMAND work station.
ASSEMBLY UNIT	Source code that can be separately assembled to produce object code. In Assembler H, a source module (see below).
ASSIGNED BEACON	The mode 3/A beacon code assigned to a track CODE by the computer or controller.
ASSOCIATED TRACK	A track that is actively correlating with a target and which has either manually or automatically been associated with a flight plan.
AVAILABLE WRITE TIME	The amount of processing time available for the system to output alphanumeric radar data to the displays.
AZIMUTH CHANGE PULSE	360 degrees is divided into 4096 equal parts and thus each azimuth change pulse (ACP) equals .088 degrees.
BASE	A number base; quantity used implicitly to define some system of representing numbers by positional notation; radix.
BASELINE	The official version of a product; used on the New York TRACON demonstration to mean the software delivered by FAA to DTC at the start of the contract.
BATCH MODE	Execution of an offline program non-interactively.
BEACON REPORT	An aircraft target detection message which is formed in the BDAS and sent to the BDAS correlation logic for possible correlation with a radar report.
BIN	A gate formed around a track's predicted position which is used to correlate the track with a target report.
BINARY	A characteristic, property, or condition in which there are but two possible alternatives; e.g., the binary number system using 2 as its base and only the digits zero (0) and (1).
BINARY-CODED DECIMAL (BCD)	Representing decimal digits with binary digits.
BIT	A single character in a binary number.

BLOCK	A unit of Pascal/VS source code consisting of an optional declaration section followed by a statement section.
BRITE ALPHANUMERIC DISPLAY	A display of sufficient brightness for presenting radar and/or other data in a control tower.
BUFFER	A temporary storage area where information is held for transfer to/from internal or external storage.
BUILD	An integral software system that contains a subset of (or all of -- if it is the final build) the system capabilities. A build represents the last step in integrating software source units. The final software system may be the product of several builds, each one built on top of the previous.
CENTRAL TRACK STORE	The major internal table containing data for each track.
CLEAR	To restore a storage or memory device to the zero state. COMPILATION UNIT Source code that can be separately compiled to produce object code. (In Pascal/VS, a SEGMENT or a PROGRAM.)
COMPILER	A computer program which produces machine language instructions and subroutines from source statements.
CONCURRENT DESIGN	The rules that describe how subtasks (see below) interact as they compete for system resources (processor, channels, data, etc.).
CONSOLIDATION	Combine control positions. This entry will transfer the control of all present and future assigned flight plans from one control position (b) to another designated, surrogate, control position (a). This includes store tracks, flight plans assigned by keyboard symbol, and flight plans assigned by fix pairs. In addition, handoffs directed to a combined position (b) will be transferred to the surrogate position.
CONTINUOUS DATA RECORDING (CDR)	Extraction of operational data on disk or tape.
CONTINUOUS DATA RECORDING EDITOR	Function to reduce operational data collected during CDR run, with the capability of filtering CDR data.

CONTROL SECTION	The smallest subdivision of an Assembler program that can be relocated as a unit. Each control section is assembled as part of an object module. By writing the proper linkage edit control statements, you can select a complete object module or any individual control section of the object module to be linkage edited and later loaded as an executable program.
CONTROLLER	A keyboard action from the DEDS. COMMAND
CONTROLLER POSITION	A keyboard status that has full keyboard entry capability (except for supervisory) and can have tracks assigned to it.
CONTROLLER SECTOR	An area of real airspace mapped onto display coordinates that composes an application work station.
CONVERSATION	Two-way communication between processes.
COPY	Instruction to the Assembler to get additional source lines from a macro library. The Assembler will replace the copy statement with a predetermined list of source statements from a library.
CORRELATION	The process whereby target data are uniquely identified with a given track.
CRITICAL DATA	Operational data which is recorded periodically by the operational program for use during a system recovery.
CROSSTELL	The point at which the status of a track during handoff is indeterminate.
CYLINDER	A set of tracks on disc pack which have in common the same track number.
DATA TYPE	The specification of a set of related data (objects) and the legal operations on that data (for example, an integer, a queue or an operating system access method.)
DEVICE, INPUT	A unit designed to bring data to be processed into a computer, e.g., a tape reader or a keyboard.
DEVICE, OUTPUT	A unit designed to output processed data from the computer, e.g., display.
DIGITIZE	To convert an analog measurement of a physical variable into a numerical value, thereby, expressing the quantity in digital form.

DISC PACK	A storage device consisting of a stack of rotating magnetic discs which are used to store and recover digital data. The disc pack is used on a disc drive.
DISCRETE CODE	A unique train of electronic pulses transmitted by an aircraft transponder in reply to a radar beacon interrogator. A four digit code in which one or both of the last two digits is other than zero.
DISPLAY CONSOLE	A visual display unit which can display information.
DOMAIN	The set of input values that can cause a function to execute.
DOWN TIME	The time during which a computer is malfunctioning during scheduled hours of operation, i.e., power failure.
DUMP	To transfer all or part of the contents of one section of the computer memory into another section or type of storage.
EDIT	The off-line process that can reduce and selectively filter the data that was extracted by the operational program and provide an operator usable output on a printout device.
ENTRY POINT	A location in a module to which control can be passed from another module or from the control program.
EXECUTION UNIT	Object code that can be executed on a computer. (A load module).
EXECUTIVE ROUTINE	A routine which directs execution of other routines in concurrent design, the subtasks (see below).
FAILSAFE	A system in which failure recovery is possible from single point failures without loss of system capability.
FAILSOFT	A system in which failure recovery is possible from single point failures where system operation continues in a degraded mode.
FAULT	A condition under which a malfunction occurs causing an interruption of the processor. This malfunction may have been caused by a physical breakdown or the attempted execution of an illegal function code.
FIELD	A set of one or more characters which is treated as a whole, a unit of information.

FILTER	A computer software function which performs the task of specifying which parts of another computer word or data word that are to be operated on or interpreted.
FLAG	A data bit used for indicating purposes or for a status condition.
FLIGHT	The set of data that defines and characterizes an aircraft controlled within ARTS.
FLIGHT PLAN STATUS	A track file in Central Track Store that is not yet active, and has not met specific time criteria for display in the tabular list or designation of flight plans being IFR, VFR or OTP.
FORCE	To intervene in a routine, by means of normal operation or programmed operation and change the normal sequence of computer operations.
FROZEN FULL DATA BLOCK	A condition resulting when a track in active handoff coast has a firmness of zero or the coast count parameter (CCNTQ) value has been reached and, for interfacility handoffs, subsequent Track Update (TU) messages are interrupted.
FRUIT	Beacon transponder replies received by the radar which are the result of interrogation from another radar. These replies are asynchronous with the radar's timing circuits and can therefore be discriminated against.
FULL CONSOLIDATION	Combine control positions and transfer all tracks. This entry will accomplish all functions of a consolidation and also, transfer tracks.
FUNCTION	A mathematical (Cartesian) function; a set of ordered pairs -- f(x,y); an algorithm modeled as a function; used during implementation to mean a Pascal/VS function that, given an argument, returns a value; for example, "square-root". See also the definition for rules.
GARBLE	The superimposing of a set of code pulses on either another set of code pulses or on noise, so that it cannot be deciphered.
GATEWAY	A set of programs and data that provide the concurrent interface control for a package.

HANDOFF STATUS — Refers to a track which is in the process of having responsibility for its control passed from one control position to another controller or from one facility to another.

HARDWARE — The mechanical, electrical, magnetic and electronic devices from which a computer or peripheral device is constructed.

HIT — A given response to an interrogation.

IDENTIFIER — A name.

ILLEGAL CHARACTER — A character which will not be accepted as a valid representation of data or function.

INITIALIZE — To set counters, addresses or switches to zero or other starting values at the beginning of a routine or program.

INPUT — Information transferred from auxiliary or external storage into core storage of the computer.

INPUT/OUTPUT — A means of communication between a computer and external equipment of other computers.

INSTRUCTION — A computer word which is a coded directive to the control section to initiate a prescribed sequence of steps necessary to effect a particular logical operation.

INTERACTIVE (I) — A process initiated by an external input that may interrupt and modify the flow of data through the system.

INTERACTIVE APPLICATION WORK STATION — The device (the DEDS) (including hardware and software), functions and people that interact with the system to create, observe, modify or delete system application work.

INTERROGATOR — Ground equipment to generate mode interrogations which trigger airborne beacon transponders and receiver responses therefrom.

INTERFACE — A common boundary between parts of a single automatic data processing system or between entire systems.

INTERLACE — The specified sequence of mode interrogations, on a sweep-to-sweep basis, used by a given beacon system.

INTERRECORD GAP — The unrecorded portion between records on magnetic tape.

INTERRUPT — A manually or automatically generated request, detected by the IOP, that a specific condition exists.

KEYBOARD ENTRY — A means of entry of alphanumeric data.

LEGAL LIMITS — Sixty-three and seven eighths (63 7/8) miles (Radar) in range for the applicable subsystem.

LEVEL-1 PACKAGE — The top level software parts that compose the monitor and application software; in sequential design, the Ada packages that define the top level abstract data types.

LEVEL-2 PACKAGE — The lowest level packages that decomposes from level-1 package.

LINK EDIT — Process of combining separately compiled object modules into an executable load module. The output of a link edit is a load module; symbolic cross references among object modules are resolved during link edit.

LIMITED CONSOLIDATION — Partially combine control positions. This entry will transfer the control of specific types of flight plans from one control position (b) to another designated position (a). Those transferred are store tracks (flight plans), all flight residing in CTS, and future flight plans assigned by fix pairs. Handoffs addressed to (b) will be redirected to (a), unless a virgule is appended to the entry.

LINEAR PROTECTION — The straight line prediction of an aircraft path.

LOAD — To read information into the computer.

LOAD MODULE — A collection of programs that will execute independently; in the demonstration, a load module maps one-to-one onto a Pascal/VS main program.

LOADER — Combines the basic functions of a linkage editor with the execution of a program. Used during testing of a load module.

LOAD MODULE — Object code in a format suitable for execution; the output of a link edit.

MACRO — Instruction to the Assembler to get additional source lines from a macro library.

MACRO DEFINITION A set of statements that define the name format of, and conditions for generating, a sequence of assembler instructions from a single source statement (macro call or macro instruction).

MACRO LIBRARY A library containing macro definitions.

MAGNETIC TAPE A storage medium consisting of metal, paper or plastic tape coated with magnetic material.

MAIN PROGRAM A single-entry, single exit Pascal/VS program that executes independently; it may contain (synchronous) calls to other programs that have been compiled separately but link-edited with it. Typically, a main program is equivalent to a Level-1 package.

MAINTENANCE POSITION A keyboard status that has restricted keyboard entry capability and tracks cannot be assigned to this position unless it has been paired to another position.

MICROSECOND One millionth of a second, 10-6 seconds.

MILLISECOND One thousandth of a second, 10-3 seconds.

MODE 3/A An interrogation mode in which a beacon radar transponder automatically reports identification when interrogated by a ground station (64 and 4096 codes).

MODE C An interrogation mode in which a beacon radar transponder automatically reports altitude when interrogated by a ground station.

MODEM A device which converts digital pulses to modulated audio signals for transmission via telephone circuits and converts the received modulated audio signals back to digital pulses.

MODIFIED BUILD A version of a build that has been changed as a result of fixing an error or errors.

MODIFIED STRING A version of a string that has been changed as a result of fixing an error or errors.

MONITOR A set of programs and data that provide data and control synchronization for the software system without knowledge of the application.

MULTIPROCESSING A technique for handling numerous tasks simultaneously through the use of an executive control program and more than one processor.

MVS	Batch operating system for IBM 370 series machines.
NANOSECOND	One billionth of a second, 10-9 second.
NONDISCRETE	A unique train of electronic pulses transmitted by an aircraft transponder in reply to a radar beacon interrogator. A four digit code in which both of the last two digits are zero.
OBJECT	A variable that can be computed by a machine.
OBJECT MODULE	The output of a compiler or an assembler. (In our application, the Pascal/VS compiler and Assembler H.) Object modules are input to the linkage editor or loader.
OFFLINE	A run separate from an online run; a data processing job that runs under VM.
ON-CALL	A set of operational support programs that reside on disc and may be loaded one at a time into a program buffer area. These programs, when loaded perform system support functions simultaneously with the execution of the operational program.
ONLINE	A run (or execution) of the New York TRACON operational system. The online system runs under MVS/RTX.
OPERATIONAL MODE	Running the ARTS with live target and live controller interaction -- as it would be run in the field.
OPERATION	A visible (defined in the specification part of a package) set of rules that act on the data encapsulated in an Ada package (state data) or that define a pure function defined in that package.
OPERATIONS	Work required to run the system (hardware, software and human interface to them); the software that automates that work, such as the online operating system.
OPERATIONS MESSAGE	Units of data through which the system and the humans or machines running the system communicate.
OPERATIONAL PROGRAM	The Automated Radar Terminal System which creates a semi-automated air traffic control system and is suitable for application to terminal radar facilities with varying densities and complexities.
OPERATIONS WORK	Units of data containing system operations messages.

OPERATIONS WORK STATION	The device (including hardware and software), functions and people that interact with the system tc create, modify or delete system operations work.
OUTPUT	Computer data that is transferred from internal storage to secondary or external storage, or to any device outside the computer.
PAIRED POSITION	A keyboard position that has full keyboard entry capability, but cannot be assigned tracks.
PARAMETER	A quantity which specifies operating conditions or configurations. The descriptions of variable data and tables.
PARITY CHECK	Checking the one bits of a block of data to test whether the total number is odd or even.
PASCAL/VS LOAD MODULE	An execution unit consisting of the link edited copy of user written functions and procedures and Pascal/VS Run time routines which are automatically supplied to the programmer.
PASCAL/VS PROGRAM	Main Program. The name of the outermost procedure of the program being run. It is a self-contained and independently compilable and executable unit of code; the program that gains initial control when the load module is invoked by RTX.
PASS	One cycle of processing a body of data.
PATCH	A section of coding inserted into a routine to correct or alter the routine.
PDL	Process Design Language, a formal language used to record software design.
PDL/ADA	A PDL based on Ada. (see Ada packages; PDL/Ada packages can also -- in addition to modeling an abstract data type -- contain a set of data or a collection of related functions.)
PERIPHERAL EQUIPMENT	Various units or machines that are used combination in or conjunction with the computer but are not part of the computer itself.
PIPELINE (P)	A set of processes that execute in order; process A produces data for process B and so on.
PLANNED TASK	An operational subprogram that is scheduled periodically by a lattice.

PREDICATE	A statement that can be evaluated; eg., max (a,b).
PRIMARY RADAR	The non-active portion of the terminal radar system which utilizes radar pulse energy which has been reflected off the aircraft skin for the generation of the primary radar data.
PRETRIGGER	A pulse generated by the beacon interrogator
PULSE	which is used for timing subsequent mode 3/A and mode C pulses.
PROCEDURE	A collection of Ada statements that define the function rules for a specific operation on a package's state data (see definition of a state machine). A procedure can also be a function that is not visible (in the specification), but is an elaboration of a visible procedure (and would be defined only within the body of the package.) A procedure can also be a Pascal/VS program that can be called from within a main program; the procedure may be separately compiled.
PROCESS	The execution of a subtask that operates on a unit of application work.
PROGRAM	A computer program (a set of executable statements, data, and commentary.)
PROGRAM ENTRY POINT	Address in the load module to be given control by the control program whenever the load module is executed.
PROPOSITION	A statement that can be evaluated true or false; e.g., x is in the set A; the state, s, of a proposition can be expressed as either s(i,TRUE) or s(i,FALSE). 14
PSEUDO FULL BLOCK DATA	A term used to describe any full data block initiate for display within ARTS by an Initiate Transfer (TI) message and/or updated by positional information contained within Track Update (TU) messages received from the ARTCC and is not being predicted or tracked by ARTS. NAT (No ARTS Track) will be displayed in field two of the FDB.
PULSE REPETITION FREQUENCY	The rate at which radar or beacon interrogations are transmitted, expressed in pulses per second.
RADAR REPORT	An aircraft target detection message which is formed in the RDAS and transmitted BDAS for possible correlation with a beacon report.
RANGE	The set of output values that result from the execution of a function.

RANGE COUNTER	A counter which measures distance in 1/64 NM increments between a radar or beacon interrogation pulse and a reply.
REAL TIME CLOCK	Develops periodic signals for the computer to allow computation of elapsed time between events.
RECORD	A set of one or more consecutive fields on a related subject.
RELEASE	A completed build -- one that has successfully passed software integration and testing.
RESECTORIZATION	The selection of one of the alternate sets of predefined combinations of entry/exit fixes and controller responsibilities.
RING-AROUND TARGET	A target whose number of hits exceeds a parametric value between its leading edge declaration and its trailing edge.
ROUTINE	A set of computer instructions arranged in such a way as to solve some defined problem (program).
RTX	Control program running under MVS to provide realtime services to applications running under it.
RULE	A specification that defines the behavior of an algorithm; x :=max(a,b) replaces x with the maximum of the values defined by a and b.
RUN	One or several routines linked to form an operating unit where the operator does not need to intervene.
SCALE FACTOR	The coefficients used to multiply or divide quantities in order to convert them so they lie in a given range of magnitude.
SCAN	One full 360 degree rotation of a radar antenna.
SCOPE	Lexical scope. The area of a module where a particular identifier can be reference is the scope of that identifier. Since routines may be nested, a lexical level is associated with each routine. Record definitions also define a lexical scope for fields of the record. An identifier can be declared only once in each lexical level.

SCOPE RULES — The rules that are applied to determine the scope of an identifier. In Pascal/VS static, block-structured scoping is applied. Any identifier defined within a block, is global to any procedure within that block. If procedures are nested, the compiler will search up the hierarchy of procedures until it finds the declaration of the identifier. The identifier declared at the innermost level is the identifier that is found.

SECOND-ORDER — Application messages that modify the state MESSAGES of a track or a flight.

SECTOR — A subset of an antenna scan; there are 32 sectors per scan. A pie shaped wedge model of a radar scan which is defined by a starting azimuth and a ending azimuth. The typical sector size used in the NY TRACON system is 128 ACPs wide, or 11.25 degrees and is used as a reference measurement in real time processing programs.

SEGMENT — A shell in which procedures and functions may be separately compiled. A compilation unit. Must be link edited with a Pascal/VS program to form a load module.

SENSOR — A unique radar antenna. The A5.04 N.Y. Tracon operates in a four sensor environment.

SEQUENTIAL DESIGN — The decomposition of the software system into parts (and their relationship to one another), expressed in a deterministic way without regard for processing concurrency, data coherency and the impact of executing the software on machines.

SITE VARIABLE PARAMETERS — The parameters that are defined through adaptation by each site to meet their particular requirements. They may vary from site-to-site and can be changed by each to accommodate their specific requirements.

SMOOTH — To apply procedures that decrease or eliminate rapid fluctuations in data.

SOFTWARE — A computer program.

SOFTWARE MESSAGES — Units of data through which packages communicate.

SOURCE MODULE — A sequence of Assembler instructions that can be separately assembled. Produces a separate object module.

SPECIAL CHARACTER — A character that is neither a number nor a letter, e.g., *,$,+,/.

SPECIFICATION A precise definition of the expected behavior of an Ada package, given in terms of its objects and the operations that act on them (see the definition of an Ada package.)

STAGGERED MODE RADAR A non-constant Pulse Repetition Frequency radar.

START An Assembler instruction used to specify the first executable control section of a source module.

STATE The function of a value mapped to an identifier; thus, s(i,v), where v is the value of identifier i.

STATE MACHINE A function with memory, such that the domain is defined by a set of input values and the current state (set of values) of the memory (referred to as state data) and the range is defined by a set of output values and the new state of the memory; the state machine function is called a transition function and represents the union of possible operations on the state data (for example, an integer behaves as a state machine: the union of arithmetic operators s(+,-,/,*) defines the transition function acting on the set of whole numbers.

STORAGE A device capable of receiving data, retaining them for indefinite periods of time, and supplying them upon command.

STORE STATUS A track file in Central Track Store than is not yet active, but has met specified time criteria and is eligible for display in the tabular list.

STRING A set of functionally-related programs, such as tracking.

SUBROUTINE A portion of a routine to which control is transferred upon instruction to carry out some operation and after executing may transfer back to the main routine.

SUBTASK A set of applications software that operates independently, and is dispatchable under MVS-RTX; the binding of a work unit to a level-1 package.

SUPERVISORY POSITION A keyboard status that allows certain exclusive keyboard functions to be performed.

SUSPEND STATUS A track which has had data block display on it temporarily terminated by keyboard action.

SWEEP — One beacon radar pulse proceeding from the interrogator to the end of the range interval at one particular azimuth.

SYSTEM DATA AREA — A predefined area on the display console where general system and site information is presented in a series of alphanumeric characters.

SYSTEM VARIABLE PARAMETERS — The adapted parameters that are defined for the ATC computer system. System parameters can be changed to accommodate the requirements of the system but cannot be changed by the site.

TABLE — A collection of data that can be identified by code or uniquely placed position.

TABULAR COAST STATUS — A condition whereby tracking correlation no longer is attempted because correlation has failed for a parametric number of scans, or confidence in the predicted position for future correlation is zero.

TARGET REPORT — An aircraft target detection message which is formed in the SRAP and sent to the DPS. The three categories of target reports are Beacon Only, Radar Only, and Radar Reinforced Beacon.

TASK — The automatic execution of system work under the control of an operating system that allocates and monitors all the system resources (channels, devices, memory, programs etc) required to perform the work.

TEMPORARY — An area of memory that is used to store STORAGE transit computer data when no long term storage is desired.

TEST MODE — Running the ARTS system using simulated targets, flights, and controller commands.

TEXT FILE — A file which contains the object code created during an assembly.

THREAD — A programming technique for linking together data files by providing pointers within each file identifying the next file in the chain. An operational example would be having all CTS files that are in the same sector for the same sensor all be in one thread (TNP).

TRACK — A computer model of an aircraft's position and velocity, -- maintained in real work and display coordinates; a dynamic record of the aircraft's behavior.

TRACK ALL — A tracking scheme where all qualifying declared targets are tracked whether or not they are associated with flight data or a controller.

TRACK FILE — A track file is a computer record containing active tracking and/or flight data.

TRACK FIRMNESS — A number functionally related to the correlation history of the track. The greater the number, the more accurate the correlation. The lower the number, the lower the accuracy of the correlation.

TRANSPONDER — An airborne radar beacon receiver-transmitter which receives radio signals from an interrogator on the ground and selectively replies with a specific reply pulse sequence. (Mode A - Beacon; Mode C - Altitude)

TYPE — A specification of the operations that can be performed on a set of data.

TYPE AREAS — Airport Area Types
I = Airport Vicinity
II = Approach Vicinity
III = All else

UNASSOCIATED — A track that is actively correlating with TRACK a target but has not been associated with any flight data.

UNIT — The smallest measurable collection of source statements; typically, a Pascal/VS procedure, a macro, a function, a homologous set of data declarations.

VALIDITY TARGET CODE — A numeric value assigned by the BDAS to a Mode 3/A code or Mode C to indicate its reliability based on code reception.

VERSION — A distinct copy of a unit, string or build, that represents a modification to a previous unit, string or build.

WORK HIERARCHY — A tree of categories of work units; categories are numbered from 1 to n, where 1 is at the top of the tree.

Section 2: Acronyms And Abbreviations

ABC	Assigned Beacon Code
ACID	Aircraft Identifier
ACK	Acknowledge
ACP	Azimuth Change Pulse
ADC	Azimuth Data Converter
ADU	Azimuth Distribution Unit
ALID	Airline Identification
AM	Flight Plan Amendment Message
AMB	Ambiguous Handoff Indicator
A/N	Alphanumeric
APG	Azimuth Pulse Generator
APT	Airport Table
ARP	Azimuth Reference Pulse
ARTCC	Air route Traffic Control Center
ARTG	Azimuth, Range and Timing Group
ARTS	Automated Radar Terminal System
ASCII	American Standard Code for Information Interchange
ASR	Airport Surveillance Radar
ASR-37	Teletype Model 37 Automatic Send-Receive Console Typewriter
ASR-40	Teletype Model 40 Automatic Send-Receive Console Typewriter
ATC	Air Traffic Control
ATCBI	Air Traffic Control Beacon Interrogator
ATCRBI	Air Traffic Control Radar Beacon Interrogator
ATCRBS	Air Traffic Control Radar Beacon System
ATCT	Air Traffic Control Tower
ATIS	Automatic Terminal Information Service

AUT	Auto Offset
AWT	Available Write Time
AZC	Center Azimuth
AZT	Trailing Edge Azimuth
BAM	Binary Angular Measurement
BANS	Brite Alphanumeric Subsystem
BCD	Binary Code Decimal
BCN	Beacon
BDAS	Beacon Data Acquisition Subsystem
BEX	Beacon Extractor
BMC	Beacon Micro Controller
BPM	Break Point Module
BRITE	Bright Radar Indicator Tower Equipment
BTL	Beacon Tracking Level
CA	Conflict Alert
CCD	Configuration Control Directive
CD	Common Digitizer
CDR	Continuous Data Recording
CDRS	Continuous Data Recording Subsystem
CDT	Console Data Terminal (Model 40 Teletype)
CDTSO	Continuous Data Time Selected Output
CFG	Configuration
CGD	Computer Generated Data
CLS	Current Lateral Separation between Two Aircraft
CONS	Consolidation of Positions
CPFS	Computer Program Functional Specification
CRIT	Critical Data Record

CRSL	Cross Reference Listing
CRT	Cathode-Ray Tube
C/S	Coast/Suspend
CST	Coast Status
CTS	Central Track Store
CVT	Coordinate Validity Time CX Cancellation Message
DA	Acceptance Message
DCON	Deconsolidation of Positions
DCU	Disk Control Unit
DDU	Disk Drive Unit
DEDS	Data Entry and Display Subsystem
DM	Departure Message
DNP	Do Not Process
DOM	Display Output Message
DOP	Display Output Processing
DPS	Data Processing Subsystem
DR	Rejection Message DSG Digital Sweep Generator
DT	Data Test Message
DUPAIR	Duplicate Pair Table
DX	Retransmit Message
DZ	Current Altitude Separation between two Aircraft
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECID	Enroute Computer Identification
EM	Emergency
EOM	End of Message
ESR	Executive Service Request
ETA	Estimated Time of Arrival

ETG	Enhanced Target Generator
FAA	Federal Aviation Administration
FDB	Full Data Block
FDEP	Flight Data Entry and Printout Equipment
FDP	Flight Data Processing
FIX	Fix Table
FP	Flight Plan
FPDU	Flight Plan Disc Update
GFE	Government Furnished Equipment
GI	General Information
GMT	Greenwich Mean Time (ZULU)
GND	Ground
GSI	General Systems Information Area
HD	Handoff
HJ	Hijack
HZ	Hertz
IA	Input Acknowledge
IC	Integrated Circuit
ICA	Interfacility Communications Adapter
ID	Identification
IDA	Input Data Acknowledge
IDR	Input Data Request
IF	Interfacility
IFR	Instrument Flight Rules
I/O	Input/Output
IOP	Input Output Processor
IRG	Inter Record Gap

KIP	Keyboard Interrupt Processing
KOF	Keyboard Operational Function Processing
LDB	Limited Data Block LCON Limited Consolidation
LE	Leading Edge
LINCON	Linear Conflict Prediction
LMD	Lateral Miss Distance (e.g., distance between aircraft in XY, at point of closest approach)
LRC	Longitudinal Redundancy Check
LSB	Least Significant Bit
MAT	Monitor Tab Coast
MFMAMS	Module for Maneuvering And Maneuver Sensitive Aircraft
MHZ	Megahertz
MODEM	Modulator/Demodulator
MSAW	Minimum Safe Altitude Warning
MSP	Medium Speed Printer
MSS	Mass Storage Subsystem
MTA	Magnetic Tape Adapter
MTBF	Mean Time Between Failure
MTI	Moving Target Indicator
MTP	Bulk Store Flight Plans
NAS	National Airspace System
NCP	NAS Change Proposal
NM	Nautical Mile
NSP	Non-Standard Part
OA	Output Acknowledge
OPE	Output Parity Error
OR	Out of Radar Range for the Controlling Display
OTE	Output Timing Error

OTP	VFR On-top
PASS	Pack Associated Tracks
PDB	Partial Data Block
PDOP	Periodic Display Output Processing
PFA	Probability of False Alarm
PI	Program Improvement
PN	Probability of Noise
PPI	Plan Position Indicator
PRF	Pulse Repetition Frequency
P/S	Primary/Secondary Correlation
PTD	Proposed Time of Departure
PT	Program Trouble
PTR	Program Trouble Report
PUNS	Pack Unassociated Tracks
PUR	Process Unused Reports
QLOOK	Quick Look Processing
RALM	Recovery Alarm
RAT	Report Address Table
RBC	Reported Beacon Code
RBTL	Radar Beacon Tracking Level
RDAS	Radar Data Acquisition Subsystem
RDOP	Remote Display Output Processing
REX	Radar Extractor
RF	Radio Failure
RFDU	Reconfiguration Fault Detection Unit
RMC	Radar Micro Controller
RTC	Recovery System Library

RTCC	Remote Tower Cab Controls
SA	Suspect Aircraft SD Sector Display
SED	Slew Entry Devices
SLINK	Intersubsystem Link
SP	System Parameter
SPI	Special Position Indicator (IDENT)
SRAP	Sensor Receiver and Processor
SS	Single Symbol STAT Status
SV	Site Variable Parameter
SWABS	Software Adaptation to Beacon System
SYNC	Synchronization
TA	Track Accept
TAB	Tabular List
TABC	Tentative Assigned Beacon Code
TALT	Altitude Tracking
TB	Beacon Terminate Message
TCID	Terminal Computer Identification
TCL	Tracked Chain List
TCROSS	Tracking Cross-Referencing
TDOP	Tabular DOP
TEDC	Tracking Early Discrete Correlation
TEXEC	Tracking Control
TI	Tracking Initiate Message
TINIT	Tracking Initial/Trial Correlation
TL	Target Leading Edge
TNP	Track Number Pointer
TNT	Track Number Table

TOLV	Time of Lateral Violation
TOMA	Time of Minimum Approach
TOS	Track Oriented Smoothing
TOV	Time of Violation
TPRED	Tracking Prediction
TPSEC	Tracking Primary/Secondary Correlation
TPUR	Tracking Process Unused Reports
TR	Test Data Message
TRACON	Terminal Radar Approach Control
TRK	Track
TROUT	Track Output
TT	Target Trailing Edge Threshold
TTI	Tabular Track Index
TU	Track Update
TUD	Thread Update
TUM	Track Update Message
UF/CR	Upfeed/Carriage Return
Va	Mode 3/A Validity
Vc	Mode C Validity
VFR	Visual Flight Rules

☆U.S. GOVERNMENT PRINTING OFFICE: 1987-505-068/60221

END
DATE
FILMED
MARCH
1988
DTIC